4. LE LANGAGE SQL AVANCE

1re partie

Version 2 - Septembre 2018

Support du chapitre 9, *Le langage SQL avancé*
de l'ouvrage *Bases de données*, J-L Hainaut, Dunod 2018.

Les versions actuelles de SQL couvre une plus grande gamme de fonctions que celle qui a été présentée jusqu'ici.

La plupart des extensions font partie du standard SQL:1999 communément appelé SQL3 (le standard SQL:2003 est sorti mais ses aspects spécifiques ne sont pas implémentés de manière généralisée).

Nous étudierons les extensions suivantes :

- **les vues et le contrôle d'accès;**
- **les prédicats, les procédures SQL et les déclencheurs;**
- **les nouvelles formes de la requête d'extraction de données;**
- **les extensions objet du langage SQL;**
- **le catalogue d'une base de données et les interfaces SQL des programmes d'application;**
- **l'information incomplète**

4. LE LANGAGE SQL AVANCE (1)

Contenu

4.1 Le contrôle d'accès

Contenu

a) Principes

b) Accorder un privilège

c) Retirer un privilège

d) Les rôles

4.1 Le contrôle d'accès - Principes

SQL permet de préciser les autorisations d'accès aux données. Une autorisation s'appelle un privilège. Un privilège précise un objet et les opérations qu'il est possible d'exécuter sur cet objet.

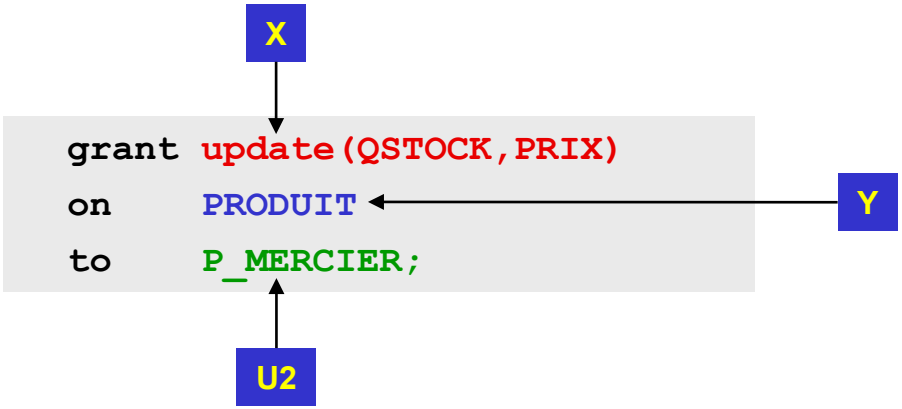
Principes :

- **l'utilisateur créateur d'un objet (table, vue, index, procédure, etc.) en devient le propriétaire; il peut effectuer sur cet objet toutes les opérations techniquement permises;**
- **le propriétaire d'un objet peut accorder à un autre utilisateur un privilège sur cet objet pour certaines opérations;**
- **le propriétaire d'un objet peut autoriser l'utilisateur auquel il a accordé un privilège à transmettre tout ou partie de celui-ci à un troisième utilisateur;**
- **le propriétaire d'un objet peut retirer un privilège accordé.**

SQL implémente le modèle de contrôle d'accès discrétionnaire.

4.1 Le contrôle d'accès - Principes

l'utilisateur U1 ← moi ...
autorise l'utilisateur U2
à effectuer l'opération X
sur l'objet Y



4.1 Le contrôle d'accès - Accorder

```
grant select, update(QSTOCK,PRIX)
on    PRODUIT
to    P_MERCIER, S_FINANCIERS;
```

```
grant select(NCLI,NOM,LOCALITE)
on    CLIENT
to    public;
```

```
grant all privileges
on    CLIENT
to    P_MERCIER, S_FINANCIERS
with  grant option;
```

4.1 Le contrôle d'accès - Accorder

```
grant run
on    SUP_DETAIL
to    public;
```

```
grant run
on    COMPTA01
to    S_FINANCIERS
with  grant option;
```

4.1 Le contrôle d'accès - Retirer

```
revoke update(PRIX)
on     PRODUIT
from   P_MERCIER;
```

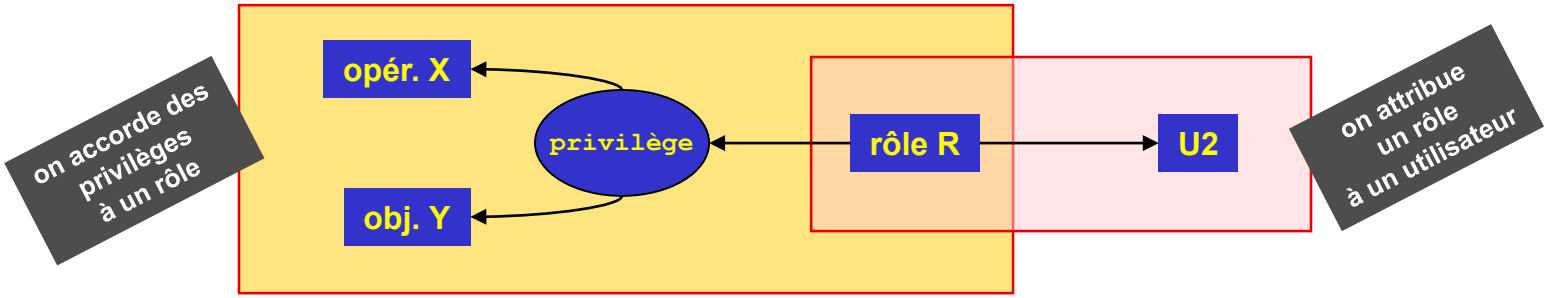
```
revoke run
on     COMPTA01
from   P_MERCIER;
```

```
revoke grant option for update (COMPTE)
on     CLIENT
from   P_MERCIER;
```

(quid si P_MERCIER avait déjà transmis ce privilège ? Cf. clés étrangères : mode *cascade* ou mode *restrict*)

4.1 Le contrôle d'accès - Les rôles

Les rôles facilitent la gestion des grands systèmes.




```
create role CONSULTANT;
```

```
grant select on CLIENT to CONSULTANT;
grant update (ADRESSE, LOCALITE) on CLIENT to CONSULTANT;
```

```
grant CONSULTANT to P_MERCIER;
```

```
revoke select (LOCALITE) from CONSULTANT;
revoke CONSULTANT from P_MERCIER;
drop role CONSULTANT;
```

4.2 Les vues SQL

Contenu

a) Définition

b) Usage

c) Modification des données

4.2 Les vues SQL - Définition

Une vue est une présentation particulière (d'une partie) des données de la base de données.

Une vue se présente comme une table virtuelle sans données stockées : ses données n'existent que lorsqu'on demande à les consulter . . .

En SQL :

```
create view COM_COMPLETE(NCOM,NCLI,NOMCLI,LOC,DATECOM)
as   select NCOM,COM.NCLI,NOM,LOCALITE,DATECOM
     from   CLIENT CLI,COMMANDE COM
     where  COM.NCLI = CLI.NCLI;
```

Quel espace une vue occupe-t-elle sur disque ?
Rien ou presque : seule la définition ci-dessus est stockée.

4.2 Les vues SQL - Définition

Une vue se manipule comme une table normale, du moins en consultation :

```
select NOMCLI,NCOM,DATECOM
from   COM_COMPLETE
where  LOC = 'Toulouse';
```

```
select NPRO
from   DETAIL
where  NCOM in ( select NCOM
                 from   COM_COMPLETE
                 where  LOC = 'Toulouse');
```

```
select LOC, count(*)
from   COM_COMPLETE CC, DETAIL D
where  CC.NCOM = D.NCOM
group by LOC;
```

4.2 Les vues SQL - Définition

Comment le SGBD exécute-t-il une requête SFW comportant une vue ?

En théorie (pour comprendre) :

le contenu de la vue est évalué et stocké temporairement sur le disque; puis la requête est exécutée sur cette table temporaire.

En pratique cette procédure est souvent trop coûteuse :

le SGBD reformule la requête SFW en y remplaçant le nom de la vue par sa définition(1) puis l'exécute.

```
select NOMCLI,NCOM,DATECOM from COM_COMPLETE where LOC = 'Toulouse';
```

⇓

```
select NOM,NCOM,DATECOM from CLIENT CLI, COMMANDE COM
where  COM.NCLI = CLI.NCLI and LOCALITE = 'Toulouse';
```

(1) En réalité le processus peut être un peu plus complexe.

4.2 Les vues SQL - Définition

On peut définir une vue sur des tables de base ou sur d'autres vues.

On peut supprimer une vue :

```
drop view COM_COMPLETE
```

4.2 Les vues SQL - Usage

A quoi servent les vues ?

1. **Interface pour des besoins particuliers**
2. **Mécanisme de contrôle d'accès**
3. **Mécanisme d'évolution de la base de données**
4. **Formulation de requêtes complexes**
5. ***etc***

4.2 Les vues SQL - Usage

1. Interface pour des besoins particuliers

But **: présentation des données adaptées aux besoins spécifiques d'un utilisateur.**

```
create view HABITUDE_ACHAT(LOCALITE,NPRO,VOLUME)
as   select LOCALITE,P.NPRO,sum(QCOM*PRIX)
     from   CLIENT CLI,COMMANDE COM,DETAIL D,PRODUIT P
     where  COM.NCLI = CLI.NCLI
     and    D.NCOM = COM.NCOM
     and    P.NPRO = D.NPRO
     group by LOCALITE, P.NPRO;
```

4.2 Les vues SQL - Usage

2. Mécanisme de contrôle d'accès

But **: limiter la visibilité des données pour un utilisateur.**

```
create view ANALYSE(LOCALITE, CAT, DATE, NPRO, QCOM) as
select LOCALITE, CAT, DATECOM, NPRO, QCOM
from   CLIENT C, COMMANDE M, DETAIL D
where  C.NCLI = M.NCLI and M.NCOM = D.NCOM;
```

```
revoke select on CLIENT from MERCIER;
revoke select on COMMANDE from MERCIER;
revoke select on DETAIL from MERCIER;
grant  select on ANALYSE to MERCIER;
```

4.2 Les vues SQL - Usage

3. Mécanisme d'évolution de la base de données

But **: protéger un utilisateur (ou les programmes d'application) des effets des modifications de structure de la base de données.**

Exemple**. Si on remplace la table :**

```
CLIENT(NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE);
```

par les deux tables complémentaires :

```
CLIENT_SIG(NCLI,NOM,ADRESSE,LOCALITE);
CLIENT_COM(NCLI,CAT,COMPTE);
```

on peut conserver les anciennes requêtes grâce à la vue :

```
create view CLIENT(NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE) as
select CS.NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE
from   CLIENT_SIG CS, CLIENT_COM CC
where  CS.NCLI = CC.NCLI;
```

4.2 Les vues SQL - Usage

4. Formulation de requêtes complexes

But **: formuler certaines requêtes qui ne peuvent s'exprimer en une seule fois. Par exemple, une fonction agrégative sur le résultat d'une fonction agrégative n'est pas autorisée :**

```
create view VAL_STOCK_ACTU(STOCK,VALEUR) as
select P.NPRO, (QSTOCK - sum(D.QCOM))*PRIX
from   DETAIL D, PRODUIT P
where  D.NPRO = P.NPRO
group by P.NPRO, QSTOCK, PRIX;

select sum(VALEUR) from VAL_STOCK_ACTU;
```

Remarque **: les limites de formulation existent surtout en SQL2. SQL3 offre plus de souplesse et de régularité.**

4.2 Les vues SQL - Modification des données

Modifier les données d'une vue (1)

Peut-on demander la modification (insert, delete, update) des données d'une vue ?

En théorie :

oui pour autant que le SGBD puisse répercuter sans ambiguïté la demande sur les données réelles dans des tables de base, mais les conditions de modifiabilité sont théoriquement complexes;

4.2 Les vues SQL - Modification des données

Modifier les données d'une vue (2)

En pratique le SGBD impose des limitations très fortes sur les vues modifiables (*updatable views*). Par exemple :

- la vue ne contient pas les clauses *distinct*, *group by*, *union*, *except*, *intersect*;
- la vue ne contient pas de fonctions agrégatives;
- la vue ne contient pas de sous-requête qui cite la table de la vue;
- la vue contient l'identifiant primaire de la table de base;
- la vue ne contient pas de jointure;
- si la vue est définie sur une autre vue, celle-ci doit être elle-même modifiable.

4.2 Les vues SQL - Modification des données

Modifier les données d'une vue (3)

Il est possible d'effectuer dans une vue une modification qui reste invisible !

```
create view CLIENT_TOULOUSE(NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE) as
select NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE
from   CLIENT where LOCALITE = 'Toulouse';
```

La modification suivante est légale :

```
insert into CLIENT_TOULOUSE(NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE)
values ('Z180','CHARLIER','rue des Gaulois','Poitiers',null,0);
```

. . . bien que la nouvelle ligne soit invisible dans la vue :

```
select * from CLIENT_TOULOUSE;
```

La vue a un <u>comportement irrégulier</u> : on n'y retrouve pas ce qu'on y a inséré !

4.2 Les vues SQL - Modification des données

Modifier les données d'une vue (4)

Pour régulariser le comportement de la base de données :

```
create view CLIENT_TOULOUSE(NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE) as
select NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE
from   CLIENT where LOCALITE = 'Toulouse'
with check option;
```

La modification suivante est rejetée :

```
insert into CLIENT_TOULOUSE(NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE)
values ('Z180','CHARLIER','rue des Gaulois','Poitiers',null,0);
```

Le comportement de la vue devient régulier : on y retrouve ce qu'on y a stocké, ni plus, ni moins !

4.3 Extensions de la structure des requêtes SFW

Contenu

a) Introduction

b) Clause Select

c) Clause From

d) Les opérateurs de jointure

4.3 Extensions de la structure des requêtes SFW

La norme SQL3 lève certaines limitations de SQL2 sur la structure des requêtes. Le langage devient plus régulier :

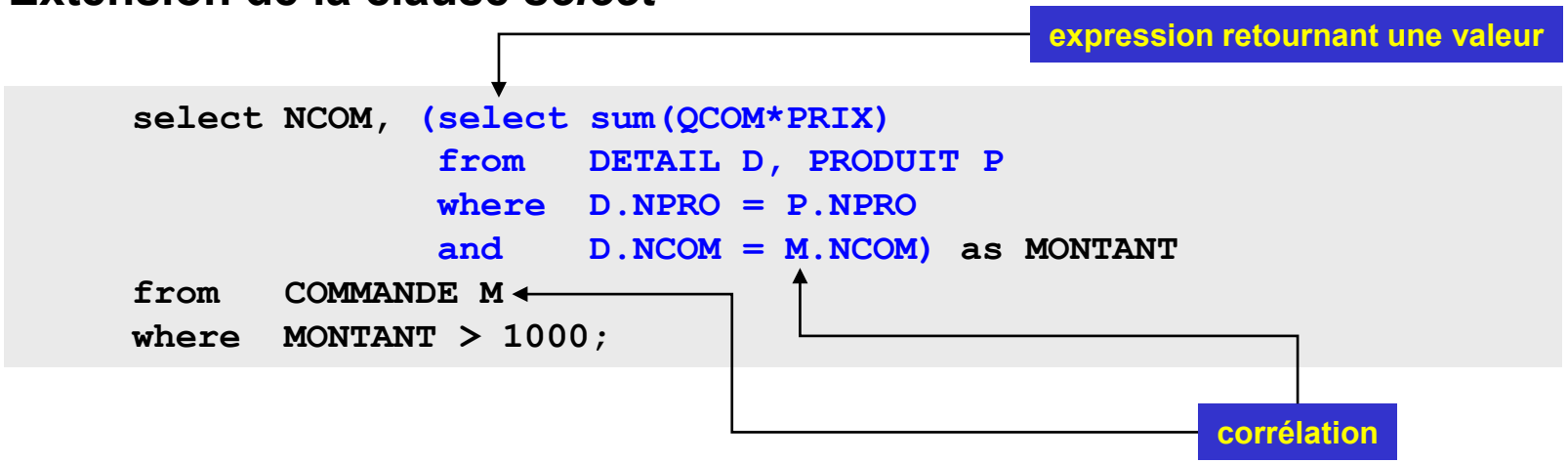
1. **là où il faut spécifier une valeur, toute expression retournant une valeur peut être utilisée (y compris une requête SFW) (*)**
2. **là où il faut spécifier une table, toute expression retournant une table peut être utilisée (y compris une requête SFW)**

La norme SQL3 introduit en outre de nouveaux opérateurs de jointure

(*) une table constituée d'une ligne et d'une colonne est considérée comme une valeur

4.3 Extensions de la structure des requêtes SFW

Extension de la clause *select*



4.3 Extensions de la structure des requêtes SFW

Extension de la clause *from*

expression retournant une table

```
select NCLI, NOM
from ((select NCLI, NOM, LOCALITE from CLIENT)
       except
      (select NCLI, NOM, LOCALITE from BON_CLIENT)
       union
      (select NCLI, NOM, LOCALITE from PROSPECT))
where LOCALITE = 'Poitiers';
```

```
select avg(MONTANT)
from  (select NCOM, sum(QCOM*PRIX) as MONTANT
       from   DETAIL D, PRODUIT P
       where  D.NPRO = P.NPRO
       group by NCOM);
```

expression retournant une table

4.3 Extensions de la structure des requêtes SFW

Extension de la clause *from*

expression retournant une table

```
select NPRO, TOTAL_QTE
from  ( (select NPRO, sum(QCOM)
         from   PRODUIT P, DETAIL D
         where  P.NPRO=D.NPRO
         group by NPRO)
         union
        (select NPRO, 0
         from   PRODUIT
         where  NPRO not in (select NPRO from DETAIL))
      )
         as DP(NPRO,TOTAL_QTE)
where TOTAL_QTE < 1000;
```

alias de cette table et de ses colonnes

4.3 Extensions de la structure des requêtes SFW

Les opérateurs de jointure (*join*)

```
select *
from   CLIENT cross join COMMANDE
where  ...
```

= produit relationnel

```
select *
from   CLIENT natural join COMMANDE
where  ...
```

= jointure naturelle

```
select *
from   CLIENT join COMMANDE using (NCLI)
where  ...
```

= jointure sur égalité

```
select *
from   CLIENT C join COMMANDE M
       on (C.NCLI = M.NCLI)
where  ...
```

= jointure sur égalité

4.3 Extensions de la structure des requêtes SFW

Les opérateurs de jointure (*join*)

```
select *
from   CLIENT C, COMMANDE M
where  C.NCLI = M.NCLI
and    ...
```

= équivalent standard

```
select *
from   CLIENT C join COMMANDE M on (C.NCLI = M.NCLI)
       join DETAIL D on (M.NCOM = D.NCOM)
       join PRODUIT P on (D.NPRO = P.NPRO)
where  ...
```

jointures multiples (peu lisibles)

```
select *
from   CLIENT C,COMMANDE M, DETAIL D,PRODUIT P
where  C.NCLI = M.NCLI
and    M.NCOM = D.NCOM
and    D.NPRO = P.NPRO;
```

= équivalent standard des jointures multiples

4.3 Extensions de la structure des requêtes SFW

Les jointures externes (*outer join*)

Pour conserver les lignes célibataires dans une jointure, on peut écrire :

```
select NCOM, C.NCLI, DATECOM, NOM, LOCALITE
from   COMMANDE M, CLIENT C
where  M.NCLI = C.NCLI
union
select null, NCLI, null, NOM, LOCALITE
from   CLIENT
where  NCLI not in (select NCLI from COMMANDE);
```

ou, plus simplement :

```
select NCOM, C.NCLI, DATECOM, NOM, LOCALITE
from   COMMANDE M right outer join CLIENT C
       on (M.NCLI = C.NCLI);
```

Egalement disponibles : `left outer join`
`full outer join`

4.4 Les requêtes récursives

4.4 Les requêtes récursives

SQL2 ne permet pas la formulation de requêtes récursives.

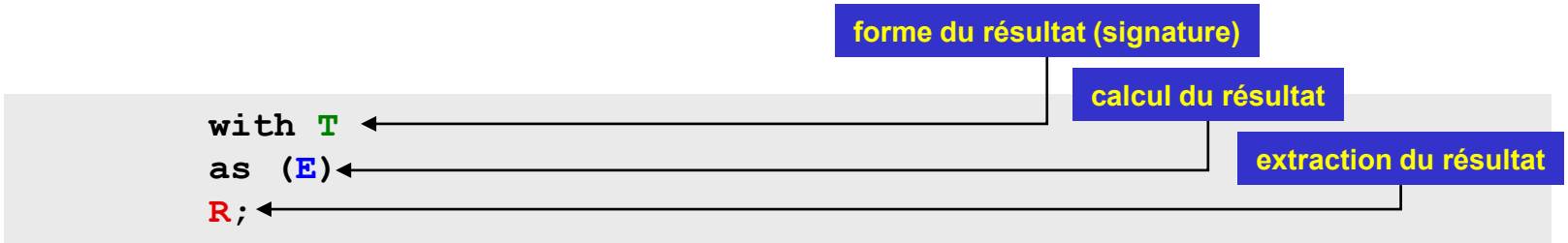
Par exemple : *lister tous les subordonnés directs et indirects de la personne p4*.

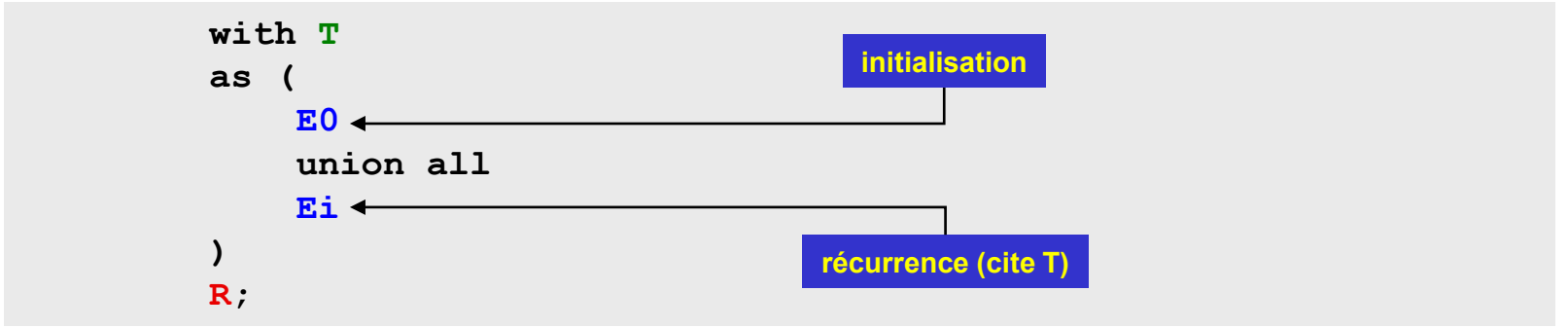
PERSONNE		
NPERS	NOM	(RESPONSABLE)
p1	Mercier	--
p2	Durant	--
p3	Noirons	p1
p4	Dupont	p1
p5	Verger	p4
p6	Dupont	p4
p7	Dermiez	p6
p8	Anciers	p2

Cette possibilité existe en SQL3

4.4 Les requêtes récursives

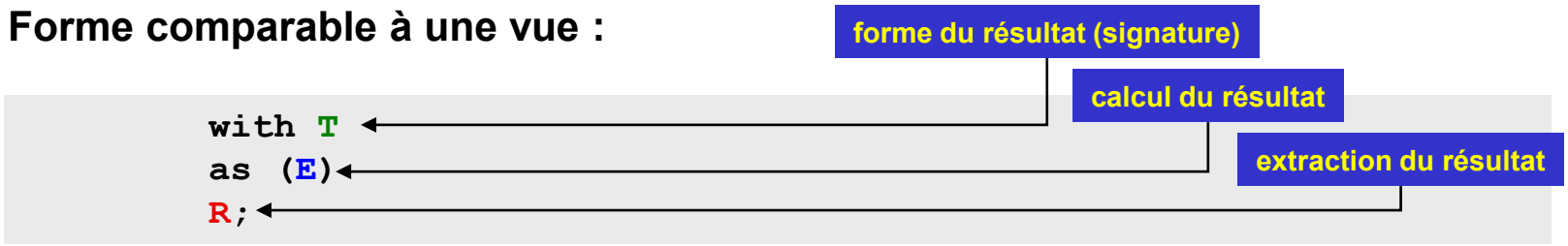
Forme générale d'une requête récursive



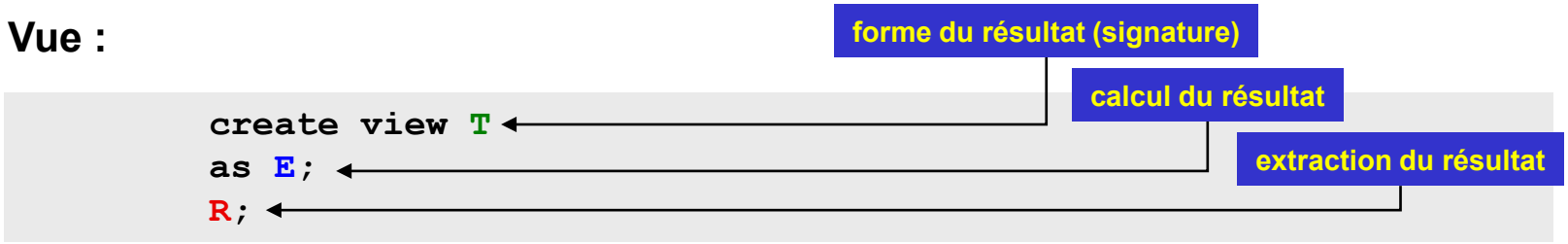


4.4 Les requêtes récursives

Forme comparable à une vue :



Vue :



Remarque : la forme *with T as* définit une vue locale à une requête, appelée *Common Table Expression*.
Elle n'est pas limitée à l'expression des requêtes récursives.

4.4 Les requêtes récursives

Exemple : *lister tous les subordonnés directs et indirects de la personne p4*

```
with ORGAN (NIVEAU, NPERS, NOM, RESP)
as (
-- Initialisation (E0)
    select 1, NPERS, NOM, RESPONSABLE
    from   PERSONNE
    where  NPERS = 'p4'
    union all
-- Incrémentation (Ei)
    select O.NIVEAU + 1, P.NPERS, P.NOM, P.RESPONSABLE
    from   ORGAN O, PERSONNE P
    where  P.RESPONSABLE = O.NPERS
)
-- Elaboration du résultat
select NIVEAU, NPERS, NOM, RESP from ORGAN;
```

LEVEL	NPERS	NOM	RESPONSABLE
1	p4	Dupont	p1
2	p5	Verger	p4
2	p6	Dupont	p4
3	p7	Dermiez	p6

4.5 Les extensions objet de SQL3

Contenu

a) Types de données complexes

b) Types définis par l'utilisateur (TDU)

c) Tables typées

d) Hiérarchies de types et de tables

4.5 Les extensions objet de SQL3

SQL3 offre de nouvelles structures de données propres au modèle de données orienté objet. Deux défis :

- **cohabitation de deux approches a priori incompatibles**
- **préserver l'interprétation SQL2**

1. **les types de données complexes : *row* et *array***
2. **les types définis par l'utilisateur**
3. **les tables typées**
4. **les hiérarchies de types et de tables**

4.5 Les extensions objet de SQL3

Types de données complexes (row)

```
create table CLIENT(
   NCLI    char(10) not null primary key,
   NOM     char(32) not null,
   ADRESSE row(RUE char(30), LOCALITE char(60)),
   CAT     char(2));
```

```
values('B332','MONTI',row('r. Neuve', 'Genève'),'B2')
```

```
select NCLI, NOM, ADRESSE.RUE
from   CLIENT
where  ADRESSE.LOCALITE = 'Poitiers';
```

4.5 Les extensions objet de SQL3

Types de données complexes (array)

```
create table CLIENT2(
   NCLI     char(10) not null primary key,
   NOM      char(32) not null,
   PRENOM   char(15) array(4),
   ADRESSES row(RUE char(30), LOCALITE char(60)) array(2),
   CAT      char(2));
```

```
select NCLI, NOM, PRENOM[1], PRENOM[2], ADRESSES[1].RUE
from   CLIENT2
where  ADRESSES[1].LOCALITE = 'Poitiers';
```

4.5 Les extensions objet de SQL3

Type défini par l'utilisateur (TDU)

extension de "create domain"

```
create type Chaine as varchar(60) default '?';

create type Contact as (RUE Chaine, LOCALITE Chaine);
```

```
create table CLIENT(
   NCLI    char(10) not null primary key,
   NOM     Chaine not null,
   ADRESSE Contact,
   CAT     char(2));
```

4.5 Les extensions objet de SQL3

Table typée

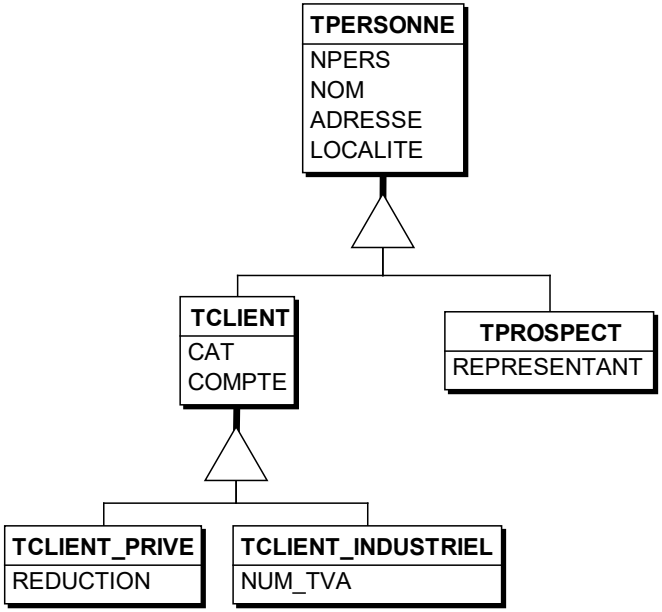
```
create type TypePERSONNE as (
   NPERS    char(10),
   NOM      Chaine,
   ADRESSE  Contact);
```

```
create table CLIENT of TypePERSONNE;

create table PROSPECT of TypePERSONNE;
```

Remarque **: les tables typées et les tables relationnelles (= classiques) peuvent coexister dans une BD**

4.5 Les extensions objet de SQL3

Hiérarchie de types



nature de l'*Object Id*

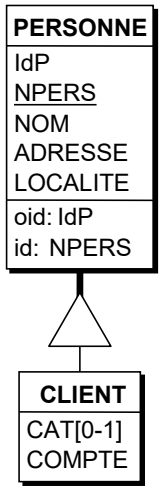
```
create type TPERSONNE as (
   NPERS     char(10),
   NOM       Chaine,
   ADRESSE   Chaine,
   LOCALITE  Chaine)
   REF is system generated;

create type TCLIENT under TPERSONNE as (
   CAT     char(2),
   COMPTE  decimal (9,2));
```

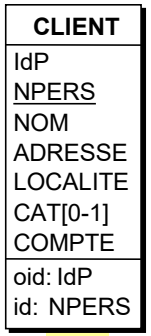
sous-type

Remarque **: les contraintes ne sont pas définies dans les types mais dans les tables.**

4.5 Les extensions objet de SQL3

Hiérarchie de tables typées





contenu *apparent* de CLIENT

```
create table PERSONNE of TPERSONNE
   (REF is IdP system generated,
    NPERS    with options not null,
    NOM      with options not null,
    ADRESSE  with options not null,
    LOCALITE with options not null,
    primary key (NPERS));

create table CLIENT of TCLIENT under PERSONNE
   (COMPTE   with options not null
                          default 0
                          check(COMPTE >= 0));
```

Object Id

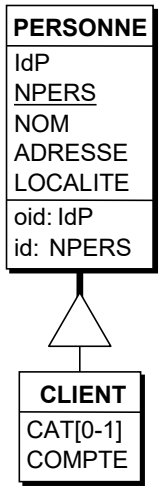
sous-table

on précise les contraintes

CAT: pas d'option à signaler

4.5 Les extensions objet de SQL3

Hiérarchie de tables typées




```
select NPERS, NOM, ADRESSE
from   CLIENT
where  LOCALITE = 'Poitiers';
```

les clients

```
select NPERS, NOM, ADRESSE
from   PERSONNE
where  LOCALITE = 'Poitiers';
```

les personnes
y compris les clients

```
select NPERS, NOM, ADRESSE
from   only(PERSONNE)
where  LOCALITE = 'Poitiers';
```

les personnes
sauf les clients

4.5 Les extensions objet de SQL3

Références entre tables

Par clés étrangères classiques ou par références (via o*bject Id*)

```
create table COMMANDE (
   NCOM char(10) not null primary key,
   DATECOM  date not null,
   REFCLI REF(TCLIENT) scope CLIENT not null);
```

```
select NCOM, DATECOM, REFCLI->NPERS, REFCLI->NOM
from   COMMANDE
where  REFCLI->LOCALITE = 'Poitiers';
```

4.5 Les extensions objet de SQL3

Compléments

- le type complexe *array* est suivi du type *multiset* en SQL:2003 (non encore stabilisé) et le sera plus tard du type *set* (déjà disponible en Oracle);
- *distinct type* : TDU simple fortement typé limitant strictement les possibilités de comparaison et opérations; objectif de sécurité de manipulation; exemple : MONTANT_DOLLARS et MONTANT_EUROS;
- type *instantiable* / *not instantiable* (= concrete/abstract en Java)
- type *final* / *not final*
- on peut associer des *méthodes* aux TDU (comme à une classe)
- notion *d'encapsulation*, de *polymorphisme* et de *surcharge*

4.5 Les extensions objet de SQL3

Le modèle relationnel objet de SQL3

- modèle hybride : relationnel + orienté objet
- mariage relationnel/objet complexe et pas toujours harmonieux : paradigmes parfois contradictoires;
- SQL3 résulte de compromis entre deux cultures différentes;
- il n'y a pas de définition rigoureuse de la simple notion de *contenu d'une table*
- exemple de l'**encapsulation** : en SQL, une requête accède aux composants d'une ligne mais en OO, la chose est interdite et ne peut se faire que via un accesseur;
  - en SQL2, l'expression select LOCALITE cite la **colonne** LOCALITE
  - en SQL3, l'expression select LOCALITE cite l'**accesseur** LOCALITE(), qui peut s'écrire plus simplement LOCALITE
  - même syntaxe, même effet mais interprétations différentes !
- standardisation encore très imparfaite; problèmes de portabilité des applications

4.6 Les prédicats

4.6 Les prédicats

Un prédicat *check* est une expression logique associée à un schéma (hélas pas implémenté), à une table ou à une colonne.

Un prédicat se présente sous la forme `check(condition)`.

La *condition* d'un prédicat doit être satisfaite à tout moment, notamment après toute modification. Si tel n'est pas le cas, la modification est annulée.

4.6 Les prédicats

Prédicat de table (1)

```
create table CLIENT ( NCLI   ...,
                      ...,
                      CAT    char(2),
              primary key (NCLI),
              check (CAT is null or
                     CAT in ('B1','B2','C1','C2'));
```

```
alter table CLIENT
add check (CAT is null or CAT in ('B1','B2','C1','C2'));
```

```
alter table CLIENT
add constraint CHK_CAT
check (CAT is null or CAT in ('B1','B2','C1','C2'));
```

contrainte nommée

```
create table CLIENT ( ...,
                      CAT    char(2),
              primary key (NCLI),
              constraint CHK_CAT check (CAT is null or
                     CAT in ('B1','B2','C1','C2'));
```

contrainte nommée

4.6 Les prédicats

Prédicat de table (2)

```
alter table COMMANDE
add check  ((DATECOM >= (select max(DATECOM)
                         from COMMANDE)
             and DATECOM <= CURRENT_DATE) is not false);
```

```
alter table COMMANDE
add check (NCLI in (select NCLI from CLIENT));
```

```
alter table CLIENT
drop constraint CHK_CAT;
```

4.6 Les prédicats

Prédicat de colonne/domaine

```
CAT char(2) check(CAT is null or
                  CAT in ('B1','B2','C1','C2'))
```

```
CAT char(2) constraint CHK_CAT check(CAT is null or
                  CAT in ('B1','B2','C1','C2'))
```

contrainte nommée

```
create domain MONTANT integer check(value >= 0);
```

***Remarque*. De nombreux SGBD limitent les *conditions* à l'état de la ligne courante. Pas question de référencer d'autres lignes, de la même table ou d'une autre table.**

4.7 Les procédures SQL

4.7 Les procédures SQL

Une procédure SQL (*Stored procedure*) est une procédure stockée dans la base de données. Elle peut être invoquée par un programme extérieur, par une procédure SQL ou par un déclencheur.

Quel est son intérêt ?

- Elle est unique pour tous les utilisateurs (simplicité de gestion et d'évolution).
- Elle a été construite par un spécialiste et est donc fiable.
- Elle peut encapsuler des fonctions complexes d'accès ou de modification de données avec validation.
- Elle peut cacher l'usage du langage SQL (*wrapping*).
- Outre la fonction de base qu'elle réalise, elle peut exécuter de manière transparente de nombreuses fonctions annexes : statistiques, journalisation, contrôle d'accès, gestion de l'intégrité des données, etc.

Malheureusement, pas de standardisation du langage des procédures SQL (langages propriétaires, Java, C#, VB, COBOL, etc.)

4.7 Les procédures SQL

Exemple : suppression *intelligente* d'une ligne de DETAIL

```
create procedure SUP_DETAIL (in COM char(12),in PRO char(15))
begin
    delete from DETAIL
    where NCOM = :COM and NPRO = :PRO;
    if  (select count(*) from DETAIL
         where NCOM=:COM) = 0
    then delete from COMMANDE
         where NCOM = :COM
    end if;
end;
```

on supprime la ligne de COMMANDE
si on vient de supprimer sa dernière ligne de DETAIL

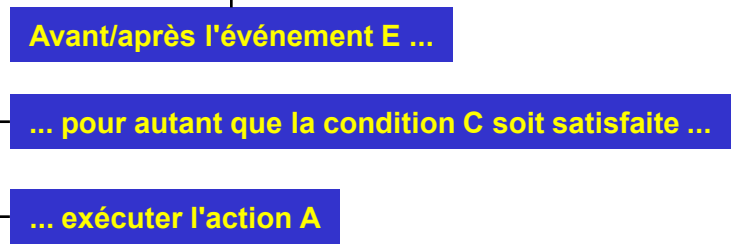
```
call SUP_DETAIL('30182','PA60');
```

4.8 Les déclencheurs

4.8 Les déclencheurs - Principes

Composant (ré)actif d'une base de données selon le modèle *événement-condition-action* (ECA)

```
before/after E
when C
begin
     A
end;
```



E est toute opération de modification des données : insert, delete, update

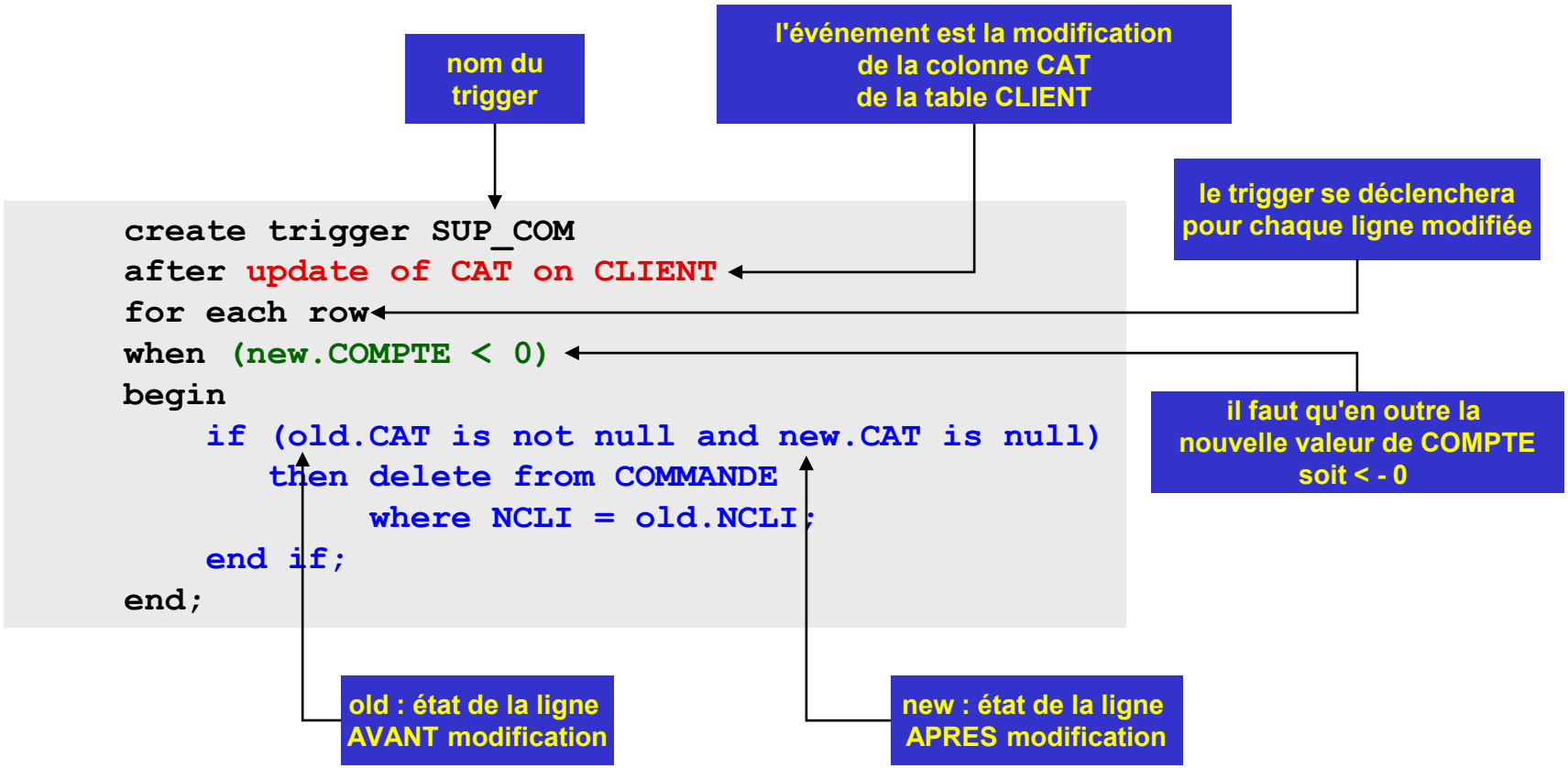
4.8 Les déclencheurs - Principes

***Exemple*. Gestion des valeurs de SUP_COM :**

```
create trigger SUP_COM
after update of CAT on CLIENT
for each row
when (new.COMPTE < 0)
begin
    if (old.CAT is not null and new.CAT is null)
        then delete from COMMANDE
             where NCLI = old.NCLI;
    end if;
end;
```

4.8 Les déclencheurs - Principes

Anatomie d'un déclencheur :




```
create trigger SUP_COM
after update of CAT on CLIENT
for each row
when (new.COMPTE < 0)
begin
    if (old.CAT is not null and new.CAT is null)
       then delete from COMMANDE
            where NCLI = old.NCLI;
    end if;
end;
```

4.8 Les déclencheurs - Principes

- Permet de donner à une base de données un **comportement actif autonome**.
- Utilisé principalement pour gérer l'intégrité des données, propager des mises à jour, gérer la redondance, implémenter des comportements spécifiques (p. ex. BD temporelles).
- Permet de construire des **bases de données actives**.
- Mécanisme très puissant mais délicat à programmer (programmation événementielle + programmation logique + programmation procédurale).

4.8 Les déclencheurs - Exemples

Exemple plus subtil (on détecte une *transition*). Lorsque la quantité en stock d'un produit devient négative, supprimer tous les détails qui en dépendent.

```
create trigger RUPTURE
after update of QSTOCK on PRODUIT
for each row
when (old.QSTOCK >= 0 and new.QSTOCK < 0)
begin
  delete from DETAIL where NPRO = old.NPRO
end;
```

La condition (new.QSTOCK < 0) serait insuffisante. Pourquoi ?

4.8 Les déclencheurs - Exemples

***Remarque*. La clause C n'est pas indispensable**

```
create trigger VERIF_PRIX1
before update of PRIX on PRODUIT
for each row
when ((new.PRIX-old.PRIX)/old.PRIX > 0.1)
begin
  raise exception 1035 'Augmentation illégale';
end;
```

```
create trigger VERIF_PRIX2
before update of PRIX on PRODUIT
for each row
begin
  if ((new.PRIX-old.PRIX)/old.PRIX > 0.1) then
  raise exception 1035 'Augmentation illégale';
end;
```

***Question*. Pourquoi l'a-t-on introduite ?**

4.8 Les déclencheurs - Exemples

Application. Simulation des *delete modes* attachés à une clé étrangère :

on delete no action

```
create trigger DEL_CLI_NO_ACTION
before delete on CLIENT
for each row
declare N integer;
begin
  select count(*) into :N from COMMANDE where NCLI = old.NCLI);
   if :N > 0 then raise exception 1042 'Erreur delete CLIENT';
   end if;
end;
```

on delete cascade

```
create trigger DEL_CLI_CASCADE
before delete on CLIENT
for each row
begin
  delete DETAIL
  where  NCOM in (select NCOM from COMMANDE where NCLI = old.NCLI);
  delete COMMANDE where  NCLI = old.NCLI);
end;
```

4.8 Les déclencheurs - Paramètres

Paramètres d'un déclencheur (1)

Evénements E affectant le contenu de la table :

insert : création d'une ligne;

delete : suppression de lignes;

update : modification des valeurs de colonnes de lignes;

autres (DDL, horloge, réseau, externes, etc) : non prévus actuellement mais proposés par certains SGBD.

Moment de l'activation du trigger par rapport à E :

before : avant l'exécution de l'événement;

after : après l'exécution de l'événement;

instead of : à la place de l'exécution de l'événement.

4.8 Les déclencheurs - Paramètres

Paramètres d'un déclencheur (2)

Portée du trigger

for each statement : trigger déclenché une fois pour chaque requête provoquant l'événement;

for each row : trigger déclenché une fois pour chaque ligne affectée par l'exécution de la requête provoquant l'événement.

Exemple : **delete from DETAIL where NPRO = 'PA60'**

fes : trigger déclenché une seule fois

fer : trigger déclenché pour chaque ligne supprimée.

4.8 Les déclencheurs - Paramètres

Paramètres d'un déclencheur (3)

Ligne(s) faisant l'objet de l'événement : 2 états accessibles (portée *fer*)

alias *old* : état de la ligne avant exécution (pour delete et update);

alias *new* : état de la ligne après exécution (pour insert et update).

Remarque : l'état *new* peut être modifié par le corps du trigger **avant** exécution de l'événement (plus possible **après**; utiliser alors un update).

```
create trigger DEL_CLI_CASCADE
before update on DETAIL
for each row
begin
  if new.PRIX < 0 then new.PRIX = 0; end if;
end;
```

4.8 Les déclencheurs - Paramètres

Paramètres d'un déclencheur (4)

Action d'un déclencheur

- annuler l'événement : si certaines conditions sont observées (violation d'une contrainte par exemple) l'événement déclencheur est inhibé ou annulé (*abort*, *raise exception*, etc.), ainsi que les actions déjà effectuées par le trigger;
- imposer des valeurs particulières à la ligne qui va être créée ou modifiée (new.CAT = 'B2');
- exécuter d'autres actions dans la base de données;
- interagir avec l'extérieur de la base de données : écrire ou lire dans un fichier externe, lancer un processus, envoyer un message, etc.

4.8 Les déclencheurs - Paramètres

Paramètres d'un déclencheur (5)

Ressources utilisables par le corps du trigger

- les états old et new : externes au trigger, déclarés implicitement;
- la base de données : externe au trigger;
- les variables du trigger : internes, déclarées explicitement.

4.8 Les déclencheurs - Remarques

Remarques

- Un trigger est attaché à une table. Il peut cependant exécuter des modifications dans d'autres tables, et ainsi déclencher d'autres triggers.
- Un trigger peut provoquer directement ou indirectement son propre redéclenchement (sorte de *récursivité*). Il importe d'étudier soigneusement de telles éventualités, et de vérifier qu'aucun événement ne risque de provoquer des chaînes infinies de déclenchements.
- Plusieurs triggers peuvent être attachés à une même table; dans ce cas, on précise l'ordre dans lequel ils sont déclenchés.
- Chaque SGBD impose certaines restrictions sur les fonctionnalités des triggers.
- Un trigger *instead of* est en général associé à une table virtuelle, c'est-à-dire à une vue. Il permet notamment de préciser l'effet des modifications dans une vue pour pallier les limites du SGBD.
- Un trigger peut exécuter des procédures SQL.
- Pas de standardisation de la syntaxe des triggers.
- Les relations entre transactions et triggers peuvent être complexes.

Fin de la 1re partie du module 4

Suite :

4. Le langage SQL avancé (2)