

# 6 Gestion des contraintes d'intégrité en SQL

## ■ *Contrainte d'intégrité statique*

- respectée pour chacun des états de la BD
- mécanismes déclaratifs
  - PRIMARY KEY, UNIQUE, NOT NULL, DOMAIN, FOREIGN KEY, CHECK, ASSERTION
- procédural
  - TRIGGER (SQL:1999)

## ■ *Contrainte d'intégrité dynamique*

- contrainte sur changements d'états
- référence aux états successifs
- TRIGGER, REFERENCES ON DELETE..., ON UPDATE ...

# 6.1 Contrainte de domaine

- Types SQL
  - INTEGER
  - CHAR
  - ...
- NOT NULL
- CHECK
- CREATE DOMAIN



## 6.1.1 Contrainte NOT NULL

```
CREATE TABLE Client
(noCLIENT      INTEGER      NOT NULL,
 nomClient     VARCHAR(15)   NOT NULL,
 noTéléphone   VARCHAR(15)   NOT NULL
)
```

- Par défaut : NULL

## 6.1.2 Contrainte CHECK sur une colonne

- Le *noClient* est supérieur à 0 et inférieur à 100,000

```
CREATE TABLE Client
(noCLIENT          INTEGER          NOT NULL
    CHECK(noClient >0 AND noClient < 100000),
nomClient          VARCHAR(15)      NOT NULL,
noTéléphone        VARCHAR(15)      NOT NULL
)
```

```
CREATE TABLE Client
(noCLIENT          INTEGER          NOT NULL,
nomClient          VARCHAR(15)      NOT NULL,
noTéléphone        VARCHAR(15)      NOT NULL,
CHECK(noClient >0 AND noClient < 100000)
)
```

## 6.1.3 Création d'un domaine (CREATE DOMAIN)

```
CREATE DOMAIN domaineSexe AS  
    CHAR(1) CHECK(VALUE IN ( 'M', 'F' ))
```

```
CREATE TABLE Employé  
(codeEmployé    INTEGER,  
...  
    sexe         domaineSexe,  
...  
);
```

```
CREATE TABLE Client  
(noClient       INTEGER,  
...  
    sexe         domaineSexe,  
...  
)
```

Pas Oracle...



## 6.1.4 Valeur de défaut (DEFAULT)

```
CREATE TABLE Client  
(  
  noTéléphone VARCHAR(15) DEFAULT 'Confidentiel' NOT NULL  
)
```

## 6.2 Contrainte de clé primaire (PRIMARY KEY)

- La clé primaire de la table *Client* est le *noClient*

```
CREATE TABLE Client
(noCLIENT          INTEGER          PRIMARY KEY
    CHECK(noClient >0 AND noClient < 100000),
nomClient          VARCHAR(15)      NOT NULL,
noTéléphone        VARCHAR(15)      NOT NULL)
```

```
CREATE TABLE Client
(noCLIENT          INTEGER          NOT NULL,
nomClient          VARCHAR(15)      NOT NULL,
noTéléphone        VARCHAR(15)      NOT NULL,
PRIMARY KEY (noCLIENT)
)
```

# Clé primaire composée

```
CREATE TABLE LigneCommande  
(noCommande      INTEGER      NOT NULL,  
 noArticle       INTEGER      NOT NULL,  
 quantité        INTEGER      NOT NULL,  
 PRIMARY KEY (noCommande, noArticle)  
)
```



## 6.3 Autres clés uniques (UNIQUE)

```
CREATE Table Citoyen
(noAssuranceSociale      INTEGER,
 noAssuranceMaladie      INTEGER,
 noPasseport             INTEGER,
 nom                     VARCHAR(30),
 prénom                  VARCHAR(30),
 dateNaissance            DATE,
 noAssSocMère             INTEGER,
...
PRIMARY KEY (noAssuranceSociale),
UNIQUE (noAssuranceMaladie),
UNIQUE (noPasseport),
UNIQUE (prénom, noAssSocMère)
)
```

## 6.4 Contrainte d'intégrité référentielle (FOREIGN KEY REFERENCES)

- Le *noClient* de la table *Commande* fait référence à la clé primaire *noClient* de la table *Client*

```
CREATE TABLE Commande
(noCommande      INTEGER          NOT NULL,
dateCommande    DATE              NOT NULL,
noClient         INTEGER          NOT NULL,
PRIMARY KEY (noCommande),
FOREIGN KEY (noClient) REFERENCES Client(noClient)
)
```

- La table *Client* doit d'abord être créée
  - privilège REFERENCES sur *Client*
- PRIMARY KEY ou UNIQUE

## 6.4.1 Politique de gestion de la contrainte d'intégrité référentielle

### ■ Tentative de mise à jour de la clé primaire

```
CREATE TABLE Commande
(noCommande      INTEGER          NOT NULL,
dateCommande    DATE             NOT NULL,
noClient        INTEGER          NOT NULL,
PRIMARY KEY (noCommande),
FOREIGN KEY (noClient) REFERENCES Client(noClient)
)
```

```
DELETE FROM Client WHERE noClient = 10
```

### ■ Options

- NO ACTION
- CASCADE
- SET NULL
- SET DEFAULT



## 6.4.1.1 Politique NO ACTION

- Rejet d'une violation de la contrainte
- Clause de défaut

```
CREATE TABLE Commande
(noCommande      INTEGER      NOT NULL,
dateCommande    DATE          NOT NULL,
noClient        INTEGER      NOT NULL,
PRIMARY KEY (noCommande),
FOREIGN KEY (noClient) REFERENCES Client
ON DELETE NO ACTION
)
```

Table <i>Commande</i>		
noCommande	dateCommande	noClient
1	01/06/2000	10
2	02/06/2000	20
3	02/06/2000	10
4	05/07/2000	10
5	09/07/2000	30
6	09/07/2000	20
7	15/07/2000	40
8	15/07/2000	40

```
DELETE FROM Client WHERE noClient = 10 {Opération rejetée}
```

```
DELETE FROM Client WHERE noClient = 70 {Opération acceptée}
```

# Cas du UPDATE

```
CREATE TABLE Commande
(noCommande      INTEGER      NOT NULL,
dateCommande    DATE          NOT NULL,
noClient        INTEGER      NOT NULL,
PRIMARY KEY (noCommande),
FOREIGN KEY (noClient) REFERENCES Client
ON UPDATE NO ACTION
)
```

Table <i>Commande</i>		
noCommande	dateCommande	noClient
1	01/06/2000	10
2	02/06/2000	20
3	02/06/2000	10
4	05/07/2000	10
5	09/07/2000	30
6	09/07/2000	20
7	15/07/2000	40
8	15/07/2000	40

```
UPDATE Client
SET noClient = 100 WHERE noClient = 10 {Opération rejetée}
```

## 6.4.1.2 Politique CASCADE

### ■ Modification en cascade

```
CREATE TABLE Commande
(noCommande      INTEGER      NOT NULL,
dateCommande    DATE          NOT NULL,
noClient         INTEGER      NOT NULL,
PRIMARY KEY (noCommande),
FOREIGN KEY (noClient) REFERENCES Client
ON DELETE CASCADE
)
```

```
DELETE FROM Client WHERE noClient = 10
{Opération acceptée temporairement}
```

```
DELETE FROM Commande WHERE noClient = 10
{Opération déclenchée automatiquement}
```



# ON UPDATE CASCADE

```
CREATE TABLE Commande
(noCommande      INTEGER          NOT NULL,
dateCommande    DATE              NOT NULL,
noClient         INTEGER          NOT NULL,
PRIMARY KEY (noCommande),
FOREIGN KEY (noClient) REFERENCES Client
ON UPDATE CASCADE
)
```

```
UPDATE Client
SET noClient = 100 WHERE noClient = 10
{Opération acceptée temporairement}
```

```
UPDATE      Commande
SET         noClient = 100 WHERE noClient = 10
{Opération déclenchée automatiquement}
```

## 6.4.1.3 Politiques SET NULL et SET DEFAULT

```
CREATE TABLE Commande
(noCommande      INTEGER          NOT NULL,
dateCommande    DATE              NOT NULL,
noClient         INTEGER          NOT NULL,
PRIMARY KEY (noCommande),
FOREIGN KEY (noClient) REFERENCES Client
ON DELETE SET NULL
)
```

```
DELETE FROM Client WHERE noClient = 10
{Opération acceptée temporairement}
```

```
UPDATE      Commande
SET         noClient = NULL WHERE noClient = 10
{Opération déclenchée automatiquement}
```

# SET DEFAULT

```
CREATE TABLE Commande
(noCommande      INTEGER      NOT NULL,
dateCommande    DATE          NOT NULL,
noClient         INTEGER      DEFAULT 50 NOT NULL,
PRIMARY KEY (noCommande),
FOREIGN KEY (noClient) REFERENCES Client
ON DELETE SET DEFAULT
)
```

```
DELETE FROM Client WHERE noClient = 10
{Opération acceptée temporairement}
```

```
UPDATE      Commande
SET         noClient = 50 WHERE noClient = 10
{Opération déclenchée automatiquement}
```



## 6.4.1.4 Clause MATCH PARTIAL/FULL

```
CREATE TABLE DétaillLivraison
(noLivraison    INTEGER      NOT NULL,
 noCommande     INTEGER      NOT NULL,
 noArticle       INTEGER,
 quantitéLivrée  INTEGER,
 PRIMARY KEY (noLivraison,noCommande, noArticle),
 FOREIGN KEY (noLivraison) REFERENCES Livraison(noLivraison),
 FOREIGN KEY (noCommande, noArticle) REFERENCES
    LigneCommande MATCH PARTIAL
)
```

Table <i>LigneCommande</i>		
noCommande	noArticle	quantité
1	10	10
1	70	5
1	90	1
2	40	2
2	95	3
3	20	1
4	40	1
4	50	1
5	70	3
5	10	5
5	20	5
6	10	5
6	40	1
7	50	1
7	95	2
8	20	3

```
INSERT INTO DétaillLivraison(noLivraison, noCommande)
VALUES (105, 4) {Opération acceptée}
```

```
INSERT INTO DétaillLivraison(noLivraison, noCommande)
VALUES (105, 10) {Opération rejetée}
```

# Oracle

- Défaut
  - ON DELETE NO ACTION
  - ON UPDATE NO ACTION
- Supporte aussi
  - ON DELETE CASCADE
  - ON DELETE SET NULL (version 8i)



## 6.5 Autres contraintes

- CHECK au delà d'une colonne
- ASSERTION générale



## 6.5.1 CHECK intra-ligne

- Plusieurs colonnes de la même ligne
- Les *Articles* dont le *noArticle* est supérieur à 90 ont un prix supérieur à \$15.00

```
CREATE TABLE Article
(noArticle      INTEGER          NOT NULL,
description     VARCHAR(20),
prixUnitaire    DECIMAL(10,2)    NOT NULL,
quantitéEnStock INTEGER          NOT NULL DEFAULT 0,
PRIMARY KEY (noArticle),
CHECK (noArticle <=90 OR prixUnitaire > 15.0)
)
```

## 6.5.2 Check inter-ligne d'une même table

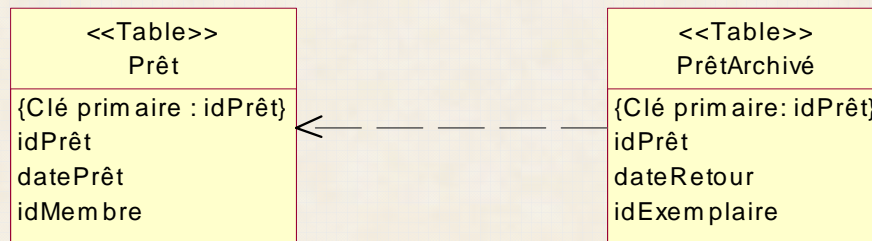
- Concerne plusieurs lignes
- Le *prixUnitaire* d'un *Article* ne peut dépasser le prix moyen de plus de \$40.00

```
CREATE TABLE Article
(noArticle      INTEGER          NOT NULL,
description     VARCHAR(20),
prixUnitaire    DECIMAL(10,2)   NOT NULL,
quantitéEnStock INTEGER        NOT NULL DEFAULT 0,
PRIMARY KEY (noArticle),
CHECK (prixUnitaire-20 <=      (SELECT AVG(prixUnitaire)
                                FROM      Article))
)
```

- Vérifié uniquement pour la ligne touchée
  - La contrainte peut être violée !
- Pas supporté par Oracle

## 6.5.3 CHECK inter-tables

- Concerne plusieurs tables



```
CREATE TABLE PrêtArchivé
(idPrêt          INTEGER          NOT NULL,
 dateRetour      DATE             NOT NULL,
 idExemplaire    INTEGER          NOT NULL,
 PRIMARY KEY (idPrêt),
 FOREIGN KEY (idPrêt) REFERENCES Prêt,
 CHECK (dateRetour >=
        SELECT datePrêt
        FROM Prêt
        WHERE Prêt.idPrêt = PrêtArchivé.idPrêt)
)
```

- Vérifié uniquement pour la ligne touchée
  - La contrainte peut être violée !
- Pas supporté par Oracle



## 6.5.4 Assertions générales

- Le *prixUnitaire* moyen d'un *Article* ne peut dépasser \$25.00

```
CREATE ASSERTION assertionPrixMoyenMaximal CHECK  
(25 >= SELECT AVG(prixUnitaire)  
FROM Article)
```

- Toujours valide par opposition au CHECK
- Non supporté par Oracle

# Assertion inter-tables

- La somme des *quantitéLivrées* pour une *LigneCommande* ne peut dépasser la *quantité* commandée

```
CREATE ASSERTION assertionQuantitéLivrée
(NOT EXISTS(
    SELECT      *
    FROM        LigneCommande AS L, DétailLivraison AS D
    WHERE       L.noArticle = D.noArticle AND
               L.noCommande = D.noCommande
    GROUP BY    L.noCommande, L.noArticle, quantité
    HAVING      quantité < SUM(quantitéLivrée)))
```



## 6.6 Implémentation de la vérification des contraintes d'intégrité

- Problème non trivial
- Vérifier uniquement les lignes modifiées
- Simplification différentielle
- Simplification sémantique
  - mécanismes d'inférence



## 6.7 Cohérence des contraintes d'intégrité

- Impossible de mettre à jour *Article* avec :

```
CREATE TABLE Article
(noArticle      INTEGER          NOT NULL,
description     VARCHAR(20),
prixUnitaire    DECIMAL(10,2)    NOT NULL,
quantitéEnStock INTEGER          NOT NULL DEFAULT 0,
PRIMARY KEY (noArticle),
CHECK (prixUnitaire > 15.0 AND prixUnitaire < 15.0))
```

- Problème difficile en général
- Aucune ou peu de vérification dans les SGBD actuels

## 6.8 Nom de contrainte (clause CONSTRAINT)

```
CREATE TABLE Client
(noCLIENT      INTEGER      NOT NULL,
 nomClient     VARCHAR(15)   NOT NULL,
 noTéléphone   VARCHAR(15)   NOT NULL,
 CONSTRAINT contNoClient CHECK(noClient >0 AND noClient < 100000)
)
```

- DROP CONSTRAINT *contNoClient*
- SET CONSTRAINT *contNoClient* ...
- Identification de la contrainte qui est violée à l'exécution

## 6.9 Contraintes différées (SET CONSTRAINTS DEFERRED)

- Quand vérifier ?
- Une *Commande* ne peut exister sans *LigneCommande* associée

```
CREATE ASSERTION assertionCommandeVide CHECK  
(NOT EXISTS  
  (SELECT *  
    FROM Commande  
    WHERE noCommande NOT IN  
      (SELECT noCommande  
        FROM LigneCommande))
```

- Vérification immédiate par défaut
  - impossible d'ajouter une *Commande* + ses *LigneCommandes* ...



# Clause DEFERRABLE

## ■ Solution au problème :

```
CREATE ASSERTION assertionCommandeVide CHECK  
(NOT EXISTS  
(SELECT *  
  FROM Commande  
  WHERE noCommande NOT IN  
    (SELECT noCommande  
      FROM LigneCommande)  
)) DEFERRABLE INITIALLY IMMEDIATE
```

```
SET CONSTRAINTS assertionCommandeVide DEFERRED
```

## ■ Ou INITIALLY DEFERRED

# 6.10 Gâchettes (TRIGGER)

- Procédure
  - déclenchée par événement pré-déterminé (INSERT, DELETE, UPDATE)
  - exécutée au niveau serveur de BD
- BD *active*
- Utilité
  - maintien de contraintes d'intégrité
    - statique
    - dynamique
    - alternative aux mécanismes déclaratifs (CHECK, ASSERTION, ...)
      - préférer mécanisme déclaratif
  - maintien d'éléments dérivés
    - colonnes dérivées
    - copies dans BD répartie
    - ...
  - historique des mises à jour
    - AUDIT
  - sécurité

## Lorsqu'une augmentation du *prixUnitaire* d'un *Article* est tentée, il faut limiter l'augmentation à 10% du prix en cours


```
CREATE TRIGGER BUArticleBornerAugmentationPrix
BEFORE UPDATE OF prixUnitaire ON Article
REFERENCING
    OLD ROW AS ligneAvant
    NEW ROW AS ligneAprès
FOR EACH ROW
WHEN (ligneAprès.prixUnitaire > ligneAvant.prixUnitaire*1.1)
BEGIN
    ligneAprès.prixUnitaire = ligneAvant.prixUnitaire*1.1;
END
```

```
UPDATE Article
SET prixUnitaire = 15.99
WHERE noArticle = 10
```

	noArticle	description	prixUnitaire
<b>ligneAvant</b>	10	Cèdre en boule	10.99

<b>ligneAprès</b>	10	Cèdre en boule	15.99
-------------------	----	----------------	-------

<b>ligneAprès</b>	10	Cèdre en boule	12.09
-------------------	----	----------------	-------





## 6.10.1 Utilisation d'un TRIGGER pour le maintien d'une contrainte d'intégrité dynamique

- Empêcher une augmentation du *prixUnitaire* d'un *Article* au delà de 10% du prix en cours

```
CREATE TRIGGER BUArticleEmpêcherAugmentationPrixTropElevée
BEFORE UPDATE OF prixUnitaire ON Article
REFERENCING
    OLD ROW AS ligneAvant
    NEW ROW AS ligneAprès
FOR EACH ROW
WHEN (ligneAprès.prixUnitaire > ligneAvant.prixUnitaire*1.1)
BEGIN
    souleverUneException;
END
```

- Oracle
  - RAISE\_APPLICATION\_ERROR

## 6.10.2 Utilisation d'un TRIGGER pour le maintien d'une contrainte d'intégrité statique

■  $0 < noClient < 100000$

```
CREATE TRIGGER BIUClientVérifierNoClient
BEFORE INSERT OR UPDATE OF noClient ON Client
REFERENCING
    NEW ROW AS ligneAprès
FOR EACH ROW
WHEN (ligneAprès.noClient <=0) OR
     (ligneAprès.noClient > 100000)
BEGIN
    souleverUneException;
END
```

■ N.B. CHECK est préférable !

## 6.10.3 Étude de cas

- Lors d'une nouvelle livraison, la quantité à livrer ne peut dépasser la quantité en stock disponible

```
CREATE TRIGGER BIDétLivVérifierQuantitéEnStock
BEFORE INSERT ON DétailLivraison
REFERENCING
    NEW ROW AS ligneAprès
FOR EACH ROW
WHEN ligneAprès.quantitéLivrée >
    (SELECT    quantitéEnStock
     FROM      Article
     WHERE     noArticle = ligneAprès.noArticle)
BEGIN
    souleverUneException;
END
```



# CHECK SQL2 inadéquat

```
CREATE TABLE DétailLivraison
(noLivraison      INTEGER      NOT NULL,
noCommande       INTEGER      NOT NULL,
noArticle        INTEGER      NOT NULL,
quantitéLivrée   INTEGER      NOT NULL,
PRIMARY KEY (noLivraison,noCommande, noArticle),
FOREIGN KEY (noLivraison) REFERENCES Livraison(noLivraison),
FOREIGN KEY (noCommande, noArticle) REFERENCES
    LigneCommande(noCommande, noArticle),
CHECK (0 <=
    SELECT    quantitéEnStock-quantitéLivrée
    FROM      Article
    WHERE     noArticle = DétailLivraison.noArticle)
)
```

**Lors d'une modification d'une *quantitéLivrée*, la différence entre la nouvelle *quantitéLivrée* et l'ancienne *quantitéLivrée* doit être inférieure ou égale à la *quantitéEnStock***

```
CREATE TRIGGER BUDétLivVérifierQuantitéEnStock
BEFORE UPDATE OF quantitéLivrée ON DétailLivraison
REFERENCING
    OLD ROW AS ligneAvant
    NEW ROW AS ligneAprès
FOR EACH ROW
WHEN ligneAprès.quantitéLivrée - ligneAvant.quantitéLivrée >
    (SELECT    quantitéEnStock
     FROM      Article
     WHERE     noArticle = ligneAvant.noArticle)
BEGIN
    souleverUneException;
END
```

# Ne permettre que la modification de la *quantitéLivrée* dans la table *DétailLivraison*

```
CREATE TRIGGER BUDétLivEmpêcherModif
BEFORE UPDATE OF noLivraison, noCommande, noArticle
ON DétailLivraison
BEGIN
    souleverUneException;
END
```



# Ajuster la *quantitéEnStock*

```
CREATE TRIGGER AIDétLivAjusterQuantitéEnStock
AFTER INSERT ON DétailLivraison
REFERENCING
    NEW ROW AS ligneAprès
FOR EACH ROW
BEGIN
    UPDATE Article
    SET quantitéEnStock = quantitéEnStock -
        ligneAprès.quantitéLivrée
    WHERE noArticle = ligneAprès.noArticle;
END
```

```
CREATE TRIGGER AUDétLivAjusterQuantitéEnStock
AFTER UPDATE OF quantitéLivrée ON DétailLivraison
REFERENCING
    OLD ROW AS ligneAvant
    NEW ROW AS ligneAprès
FOR EACH ROW
BEGIN
    UPDATE Article
    SET quantitéEnStock = quantitéEnStock -
        (ligneAprès.quantitéLivrée - ligneAvant.quantitéLivrée)
    WHERE noArticle = ligneAvant.noArticle;
END
```

# Extension procédurale pour *corpsTrigger*

- Traitements complexes
- Combiner plusieurs TRIGGER
  - vérifier quel est l'événement déclencheur
- Ordre d'exécution des TRIGGER
  - BEFORE avant AFTER,...
  - entre TRIGGER de même type ?
    - forcer un ordre en combinant
- Oracle
  - PL/SQL

## 6.10.4 TRIGGER de niveau STATEMENT

- Exécution du corps une seule fois pour plusieurs lignes mises à jours dans le même énoncé

```
CREATE TRIGGER BUDétLivEmpêcherModif
BEFORE UPDATE OF noLivraison, noCommande, noArticle
ON DétailLivraison
BEGIN
    souleverUneException;
END
```

```
REFERENCING
    OLD TABLE AS nomAvant
    NEW TABLE AS nomAprès
```



## 6.10.5 Ordre d'exécution des TRIGGER

```
Exécuter les TRIGGER BEFORE STATEMENT
Pour chaque ligne touchée par l'opération
    Exécuter les TRIGGER BEFORE ROW
    Exécuter l'opération
    Exécuter les TRIGGER AFTER ROW
Fin pour
Exécuter les TRIGGER AFTER STATEMENT
```

- Attention aux circularités !

## 6.10.6 Limites des TRIGGER

- Ne peuvent être DEFERRED
- Complexes à coder
- Contraintes particulières aux dialectes



# 6.10.7 Particularités des TRIGGER Oracle

- Pas de SELECT dans le WHEN
- :NEW, :OLD
- Omettre le mot-clé ROW dans REFERENCING
- Corps en PL/SQL (voir chapitre 4).
- Syntaxe *:nomColonne*
- Pas de COMMIT/ROLLBACK dans un TRIGGER
  - procédure PL/SQL RAISE\_APPLICATION\_ERROR
    - Intervalle [-20000, -20999] pour code d'erreur
- IF INSERTING, DELETING, UPDATING.
- Événements non standards
  - INSTEAD OF, STARTUP, LOGON, ...
- Problème avec table en mutation (modifiée par l'événement déclencheur)
- etc.



```

SQL> CREATE OR REPLACE TRIGGER BUArticleBornerAugPrix
  2 BEFORE UPDATE OF prixUnitaire ON Article
  3 REFERENCING
  4         OLD AS ligneAvant
  5         NEW AS ligneAprès
  6 FOR EACH ROW
  7 WHEN (ligneAprès.prixUnitaire > ligneAvant.prixUnitaire*1.1)
  8 BEGIN
  9         :ligneAprès.prixUnitaire := :ligneAvant.prixUnitaire*1.1;
10 END;
11 /

```

Déclencheur créé.

```

SQL> -- Test du TRIGGER BUArticleBornerAugPrix
SQL> SELECT * FROM Article WHERE noArticle = 10
  2 /

```

NOARTICLE	DESCRIPTION	PRIXUNITAIRE	QUANTITÉENSTOCK
10	Cèdre en boule	10,99	10

```

SQL> UPDATE Article
  2 SET      prixUnitaire = 15.99
  3 WHERE    noArticle = 10
  4 /

```

1 ligne mise à jour.

```

SQL> SELECT * FROM Article WHERE noArticle = 10
  2 /

```

NOARTICLE	DESCRIPTION	PRIXUNITAIRE	QUANTITÉENSTOCK
10	Cèdre en boule	12,09	10

```

SQL> CREATE OR REPLACE TRIGGER BIDétLivVérifierStock
2  BEFORE INSERT ON DétailLivraison
3  REFERENCING
4      NEW AS ligneAprès
5  FOR EACH ROW
6  DECLARE
7      laQuantitéEnStock      INTEGER;
8  -- N.B. Oracle ne supporte pas de SELECT dans le WHEN
9  -- Il faut donc utiliser un IF PL/SQL
10 BEGIN
11      SELECT  quantitéEnStock
12      INTO    laQuantitéEnStock
13      FROM    Article
14      WHERE   noArticle = :ligneAprès.noArticle;
15
16      IF :ligneAprès.quantitéLivrée > laQuantitéEnStock THEN
17          raise_application_error(-20100, 'stock disponible insuffisant');
18      END IF;
19 END;
20 /

```

Déclencheur créé.

```

SQL> -- Test du TRIGGER BIDétLivVérifierStock
SQL> SELECT * FROM Article WHERE noArticle = 10
2  /

```

NOARTICLE	DESCRIPTION	PRIXUNITAIRE	QUANTITÉENSTOCK
10	Cèdre en boule	12,09	10

```

SQL> INSERT INTO DétailLivraison
2      VALUES(105,5,10,30)
3  /

```

```

INSERT INTO DétailLivraison
      *

```

ERREUR à la ligne 1:

ORA-20100: stock disponible insuffisant

ORA-06512: à "BANQUE.BIDÉTLIVVÉRIFIERSTOCK", ligne 12

ORA-04088: erreur lors d'exécution du déclencheur 'BANQUE.BIDÉTLIVVÉRIFIERSTOCK'

# Trigger **INSTEAD OF** pour **VIEW** non modifiable (non standard **SQL:1999**)

```
SQL> SELECT * FROM Article WHERE noArticle = 10
2 /
```

NOARTICLE	DESCRIPTION	PRIXUNITAIRE	QUANTITÉENSTOCK
10	Cèdre en boule	10,99	20

```
SQL> CREATE VIEW ArticlePrixPlusTaxe AS
2   SELECT noArticle, description, prixUnitaire * 1.15 AS prixPlusTaxe
3   FROM Article
4   /
```

View created.

```
SQL> UPDATE ArticlePrixPlusTaxe
2   SET prixPlusTaxe = 23
3   WHERE noArticle = 10
4   /
```

```
SET prixPlusTaxe = 23
*
```

ERROR at line 2:

ORA-01733: virtual column not allowed here



# suite

```
SQL> CREATE OR REPLACE TRIGGER InsteadUpdate
 2  INSTEAD OF UPDATE ON ArticlePrixPlusTaxe
 3  REFERENCING
 4    OLD AS ligneAvant
 5    NEW AS ligneAprès
 6  FOR EACH ROW
 7  BEGIN
 8    UPDATE Article
 9    SET
10    noArticle = :ligneAprès.noArticle,
11    description = :ligneAprès.description,
12    prixUnitaire = :ligneAprès.prixPlusTaxe / 1.15
13    WHERE noArticle = :ligneAvant.noArticle;
14  END;
15  /
```

Trigger created.

```
SQL> UPDATE ArticlePrixPlusTaxe
 2  SET prixPlusTaxe = 23
 3  WHERE noArticle = 10
 4  /
```

1 row updated.

```
SQL> SELECT * FROM Article WHERE noArticle = 10
 2  /
```

NOARTICLE	DESCRIPTION	PRIXUNITAIRE	QUANTITÉ	ENSTOCK
10	Cèdre en boule	20		20