

## CONTROLE D'INTEGRITE SEMANTIQUE

### OBJECTIF :

Détecter les mises à jour erronées et réagir soit en rejetant la transaction, soit en compensant les erreurs.

### CECI SUPPOSE :

- un langage de définition de contraintes d'intégrité
- la vérification automatique de ces contraintes

### AVANTAGES :

- simplification du code des applications
- sécurité renforcée par l'automatisation
- mise en commun des contraintes
- cohérence globale des contraintes

## Typologie des contraintes

### CONTRAINTES D'INTEGRITE STRUCTURELLES

- Contraintes de DOMAINE  
ex: le cru d'un vin est de type chaîne de caractères
- Contraintes d'ENTITE (unicité et non nullité)  
toute relation doit posséder au moins une clé et cette clé ne peut pas prendre de valeurs nulles
- Contraintes REFERENTIELLE  
ex: l'ensemble des valeurs de l'attribut ABUS.NV doit être inclus dans l'ensemble des valeurs de l'attribut VINS.NV

### CONTRAINTES D'INTEGRITE COMPORTEMENTALES

- Contraintes générales liées à une application spécifique  
ex: la somme des quantités bues d'un vin doit être inférieure a la quantité produite de ce même vin

## Typologie des contraintes comportementales

### LES CONTRAINTES D'INTEGRITE COMPORTEMENTALES PERMETTENT DES CONTROLES TRES VARIES :

- **DOMAINE DE VARIATION**  
Ex: le degré d'un vin ne peut être inférieur à 8
- **CONTRAINTES MULTI-ATTRIBUTS (HORIZONTALES)**  
Ex: le prix de vente d'un produit doit être supérieur à son coût de production
- **DEPENDANCE FONCTIONNELLE**  
Ex : CRU, ANNEE -----> DEGRE dans la relation VINS
- **CONTRAINTES TEMPORELLES**  
Ex : le degré d'un vin ne peut pas décroître
- **CONTRAINTES AGREGATIVES (VERTICALES)**  
Ex : la somme des quantités bues d'un vin doit être inférieure a la quantité produite de ce même vin

## Typologie des contraintes comportementales

### • DEPENDANCE D'INCLUSION

Concept de généralisation

{VALEURS D'UN {GROUPE D'ATTRIBUTS X}}  
inclus dans  
{VALEURS D'UN {GROUPE D'ATTRIBUTS Y}}

EXEMPLE : ENSEIGNANT.NOM inclus dans PERSONNE.NOM  
G  
ENSEIGNANT -----> PERSONNE

### • LA DEPENDANCE REFERENTIELLE EST UN CAS PARTICULIER DE DEPENDANCE D'INCLUSION

{VALEURS DE X} inclus dans {VALEURS DE Y} et Y EST CLE

EXEMPLE : ABUS.NV inclus dans VINS.NV

## Association des contraintes

### UNE CONTRAINTE D'INTEGRITE PEUT ETRE :

- Associée à un domaine  
Spécifiée au travers de la clause CREATE DOMAIN
- Associée à une relation  
Spécifiée au travers de la clause CREATE TABLE
- Dissociées  
Spécifiée au travers de la clause CREATE ASSERTION

## Contraintes associées aux domaines

```
CREATE DOMAIN <nom> <type> [valeur]
[CONSTRAINT nom_contrainte CHECK (condition) ]
```

### Exemple:

```
CREATE DOMAIN couleur_vins CHAR(5) DEFAULT 'rouge'
CONSTRAINT couleurs_possibles CHECK
(VALUE IN ('rouge', 'blanc', 'rosé'))
```

## Contraintes associées aux relations

```
CREATE TABLE <nom_table>
(<def_colonne> *
[<def_contrainte_table>*]);
```

```
< def_colonne > ::= <nom_colonne> <type | nom_domaine >
[CONSTRAINT nom_contrainte
< NOT NULL | UNIQUE | PRIMARY KEY |
CHECK (condition) | REFERENCES nom_table (liste_colonnes) > ]
[NOT] DEFERRABLE
```

```
< def_contrainte_table > ::= CONSTRAINT nom_contrainte
< UNIQUE (liste_colonnes) | PRIMARY KEY (liste_colonnes) |
CHECK (condition) |
FOREIGN KEY (liste_colonnes) REFERENCES nom_table (liste_colonnes) >
[NOT] DEFERRABLE
```

## Contraintes associées aux relations

```
CREATE TABLE VINS
( NV INTEGER PRIMARY KEY,
couleur COULEURS_VINS,
cru VARCHAR(20),
millesime DATE,
degre INTEGER CHECK (degre BETWEEN 8 AND 15) NOT DEFERRABLE,
quantite INTEGER,

CONSTRAINT dependance_fonctionnelle
CHECK (NOT EXISTS (SELECT *
FROM VINS
GROUP BY cru,millesime
HAVING COUNT(distinct degre) > 1)
NOT DEFERRABLE) ;
```

## Contraintes référentielles

```
FOREIGN KEY (liste_colonnes)
REFERENCES nom_table (liste_colonnes)
[ON DELETE {CASCADE | SET DEFAULT | SET NULL}]
[ON UPDATE {CASCADE | SET DEFAULT | SET NULL}]
[NOT] DEFERRABLE
```

- Les contraintes référentielles caractérisent toutes les associations
- Problème des contraintes référentielles croisées ==> mode DEFERRABLE
- En cas de violation de la contrainte, la mise à jour peut être rejetée ou bien une action de correction est déclenchée ==>
  - ON DELETE spécifie l'action à effectuer en cas de suppression d'un tuple référencé
  - ON UPDATE spécifie l'action à effectuer en cas de mise à jour de la clé d'un tuple référencé

## Contraintes référentielles: exemple

```
CREATE TABLE ABUS
(NB INTEGER NOT NULL,
 NV INTEGER NOT NULL,
 date DATE,
 qte QUANTITE,
 UNIQUE (NB, NV, date)

CONSTRAINT référence_buveurs
FOREIGN KEY NB
REFERENCES BUVEURS (NB)
ON DELETE CASCADE
DEFERRABLE

);
```

## Contraintes dissociées

```
CREATE ASSERTION nom_contrainte CHECK (condition)
```

Remarque: les contraintes dissociées peuvent être multi-tables

Exemple:

```
CREATE ASSERTION quantite_produite
CHECK ((SELECT SUM(quantite) FROM VINS) >
 (SELECT SUM(quantite) FROM ABUS))
```

## Déclencheurs (TRIGGERS)

- Déclencheur : action ou ensemble d'actions déclenchée(s) automatiquement lorsqu'une condition se trouve satisfaite après l'apparition d'un événement
  - Un déclencheur est une règle ECA
- Événement = mise à jour d'une relation
- Condition = optionnelle, équivaut à une clause <WHERE>
- Action = exécution de code spécifique (requête SQL de mise à jour, exécution d'une procédure stockée, abandon d'une transaction, ...)
- De multiples usages sont possibles :
    - contrôle de l'intégrité
    - maintien de statistiques
    - mise à jour de copies multiples, ...

## Définition des triggers

```
CREATE TRIGGER <nom-trigger>
<événement>
[<condition>]
<action *>
```

<événement> ::=  
BEFORE | AFTER  
{INSERT | DELETE | UPDATE [OF <liste\_colonnes>]}  
ON <nom\_de\_table>

<condition> ::=  
[REFERENCING OLD AS <nom\_tuple> NEW AS <nom\_tuple>]  
WHEN <condition\_SQL>

<action> ::=  
{requête\_SQL [FOR EACH ROW]  
| exec\_procedure | COMMIT | ROLLBACK}

## Exemples de trigger

```
CREATE TRIGGER degré_croissant
BEFORE UPDATE OF degre ON VINS
REFERENCING OLD AS old_vin NEW AS new_vin
WHEN (new_vin.degre < old_vin.degre)
ROLLBACK
FOR EACH ROW
```

```
CREATE TRIGGER référence_vins
BEFORE DELETE ON VINS
DELETE FROM ABUS
WHERE ABUS.NV = VINS.NV
FOR EACH ROW
```

## CONTROLE DES DROITS D'ACCES

**Objectif :** protéger les données de la base contre des accès non autorisés

**Deux niveaux :**

- connexion au serveur restreinte aux usagers répertoriés (mot de passe)
- privilège d'accès aux objets de la base

**Granule d'objet :**

- vue
- relation
- procédure stockée
- trigger

## Hliérarchie des privilèges

Administrateur Système {users login, ...}

Administrateur de la Base {connexion utilisateurs à BD, gestion des schémas}

Propriétaires d'Objets {créateurs d'objets, distribution des droits}

Autres (PUBLIC) {droits obtenus par GRANT}

## Définition des droits

```
GRANT <droits> ON <objet>
TO <usagers>
[WITH GRANT OPTION]
```

<droits> ::= ALL | SELECT | DELETE |  
INSERT [<attribut>] | UPDATE [<attribut>] |  
REFERENCE [<attribut>]

<objet> ::= TABLE <relation> \* | VIEW <vue> \*

<usagers> ::= PUBLIC | <username> \*

### Exemple:

```
GRANT SELECT, UPDATE (DEGRE) ON TABLE VINS
TO DUPONT
```

## Suppression des droits

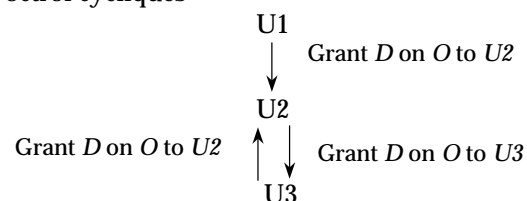
```
REVOKE <droits> ON <objet>
FROM <usagers>
```

### Exemple:

```
REVOKE ALL ON TABLE VINS
FROM DUPONT
```

## Règles d'octroi et de suppression des droits

- On ne peut transmettre que les droits que l'on possède ou qui nous ont été transmis avec "grant option"
- On ne peut supprimer que les droits que l'on a transmis
- La révocation des droits est réursive
- Si un utilisateur U1 a reçu le droit D de la part de plusieurs utilisateurs (U2, U3, ...), il ne perd ce droit que si tous les utilisateurs lui retirent
- Ces règles garantissent la cohérence des retraits même en cas de graphe d'octroi cycliques



## Autorisations sur une vue

UNE VUE EST UNE RELATION VIRTUELLE DEFINIE A PARTIR DES  
RELATIONS DE BASE PAR UNE EXPRESSION SQL

### EXEMPLE :

```
CREATE VIEW GROS_BUVEURS (NB, NOM, QTE_BUE)
AS SELECT B.NB, B. NOM, SUM(A.quantite)
FROM BUVEURS B, ABUS A
WHERE B.NB = A.NB
GROUP BY NB
```

### AUTORISATIONS SUR LES VUES:

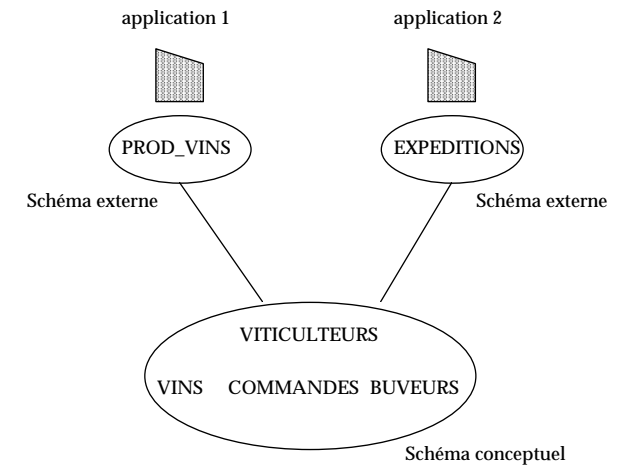
- Permet un granule d'autorisation très fin
- Cependant, les mises à jour au travers des vues sont très limitées

## LES VUES EXTERNES

LE CONCEPT DE VUES REpond AUX BESOINS SUIVANTS :

- Adaptation aux applications
- Intégration des applications existantes
- Dynamique du schéma de la base
- Confidentialité et sécurité
- Décentralisation de l'administration d'une BD
- Hétérogénéité des modèles
- Transparence à la localisation (dans le cas de bases réparties)

## Exemple de vues externes (1)



## EXEMPLE (2)

**Schema conceptuel :**

VINS (NV, CRU, ANNEE, DEGRE, NVIT)

VITICULTEURS (NVIT, NOM, PRENOM, VILLE)

BUVEURS (NB, NOM, PRENOM, VILLE)

COMMANDE (NB, NV, QTE, DATE)

on peut définir de nombreuses applications particulières qui n'utilisent qu'une partie des données du schéma.

## EXEMPLE (3)

**APPLICATION = INFLUENCE GEOGRAPHIQUE  
SUR LA CONSOMMATION DE L'ANNEE 1996**

- **VUES = RESTRICTION-PROJECTION DES RELATIONS DE BASE**

BUVEURS-1 (NB, VILLE)

ACHETE-1 (NB, NV, QTE, DATE) // restreint aux dates de l'année 96

VINS-1 (NV, CRU)

- **VUE = RESTRUCTURATION DU SCHEMA CONCEPTUEL**

ACHAT-2 (NB, VILLE, CRU, QTE, DATE)

- **VUE = RESTRUCTURATION et AGREGATION**

CONSOMMATION-PAR-VILLE

CPV (VILLE, CRU, QTE)

## DEFINITION DES VUES

```
CREATE VIEW <nom_vue> [(liste_attributs)]
AS <expression_de_sélection>
[WITH CHECK OPTION]
```

- L'expression de sélection peut porter sur des tables de base et/ou des vues
- Dans le cas de vues modifiables, la clause WITH CHECK OPTION garantit que les tuples insérés (ou modifiés) dans la vue vérifient bien le critère de la vue

## DEFINITION DES VUES: Exemple (1)

```
CREATE VIEW BUVEURS-1
AS SELECT NB, VILLE
FROM BUVEURS
```

```
CREATE VIEW VINS-1
AS SELECT NV, CRU
FROM VINS
```

```
CREATE VIEW ACHETE-1
AS SELECT NB, NV, QTE, DATE
FROM COMMANDE
WHERE DATE BETWEEN '01.01.96' AND '31.12.96'
```

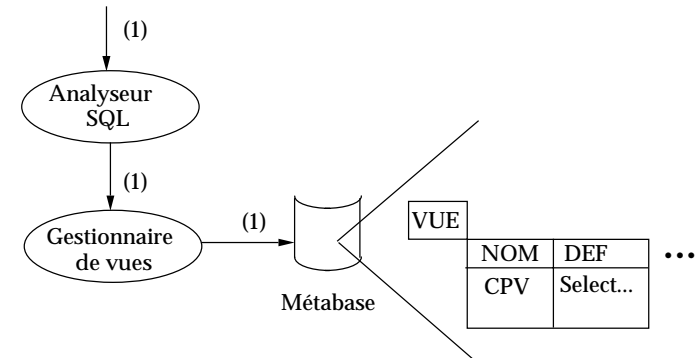
```
CREATE VIEW ACHAT-2 (NB, VILLE, CRU, QTE, DATE)
AS SELECT B.NB, B.VILLE, V.CRU, C.QTE, C.DATE
FROM BUVEURS B, VINS V, COMMANDES C
WHERE C.NB = B.NB AND
C.NV = V.NV AND
C.DATE BETWEEN '01.01.96' AND '31.12.96'
```

## DEFINITION DES VUES : Exemple (2)

```
CREATE VIEW CPV (VILLE, CRU, QTE)
AS SELECT B.VILLE, V.CRU, SUM (C.QTE)
FROM BUVEURS B, COMMANDES C, VINS V
WHERE B.NB = C.NB AND
C.NV = V.NV AND
C.DATE BETWEEN '01.01.96' AND '31.12.96'
GROUP BY B.VILLE, V.CRU
```

## DECLARATION ET INTERROGATION

```
CREATE VIEW CPV
AS SELECT...
```



## INTERROGATION DES VUES (1)

MODIFICATION DE QUESTION PAR RESTRUCTURATION SYNTAXIQUE

EXEMPLE :

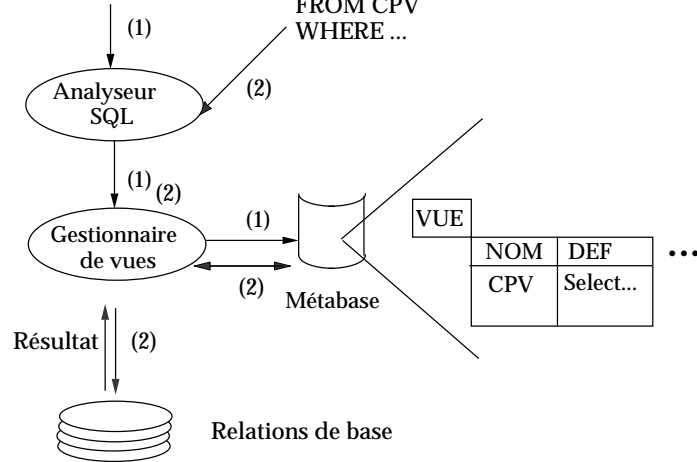
```
SELECT CRU, QTE
FROM CPV
WHERE VILLE = "PARIS"
```

DEVIENT :

```
SELECT V.CRU, SUM (C.QTE)
FROM BUVEURS B, COMMANDES C, VINS V
WHERE B.NB = C.NB AND
      C.NV = V.NV AND
      C.DATE BETWEEN '01.01.96' AND '31.12.96' AND
      B.VILLE = 'PARIS'
GROUP BY VILLE, CRU
```

CREATE VIEW CPV  
AS SELECT...

SELECT ...  
FROM CPV  
WHERE ...

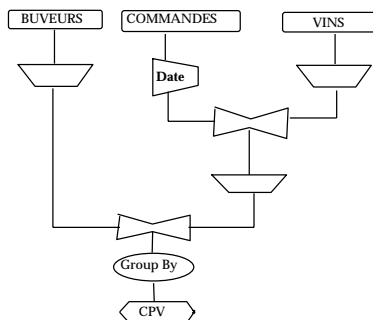


VUE	
NOM	DEF ...
CPV	Select...

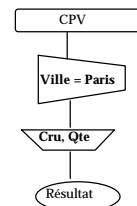
## INTERROGATION DES VUES (2)

MODIFICATION DE QUESTION PAR RESTRUCTURATION  
D'ARBRES ALGEBRIQUES

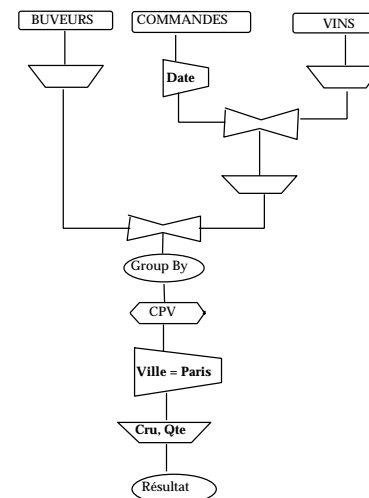
Arbre algébrique de la vue CPV



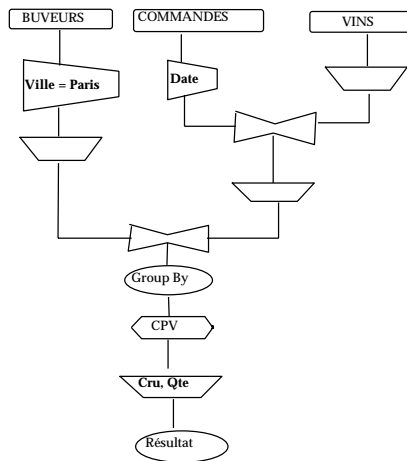
Arbre algébrique de la requête



## LES ARBRES SONT MIS BOUT A BOUT



## PUIS L'ARBRE RESULTAT EST OPTIMISE



## MISES A JOUR AU TRAVERS DES VUES

- Les mises à jour au travers des vues sont rarement définies  
Ex: Comment reporter une mise à jour sur CPV dans les relations de base ?
- Pour rendre le report de mises à jour possible, la définition de la vue doit respecter certaines contraintes:
  - la clause SELECT doit conserver les clés de toutes les relations de base
  - la clause SELECT ne doit pas contenir de calcul d'agrégat
  - la définition ne doit pas contenir de clause GROUP BY ni HAVING
- Dans SQL2, en plus des conditions précédentes, seules les vue définies à partir d'une relation unique seront considérées comme modifiables

## VUES CONCRETES

- **VUE CONCRETE** : VUE DONT ON A DEMANDE L'IMPLANTATION DANS LA BASE DE DONNEES (Virtuel ----> Réel)

INTERETS :

- INTERROGATIONS FREQUENTES,
- SYSTEMES REPARTIS

- **PROBLEMES** : MAJ DES DONNEES DE BASE A REPERCUTER SUR LA VUE

L'OBJECTIF EST D'EVITER DE REEVALUER COMPLETEMENT LA VUE A CHAQUE MISE A JOUR

## CONCLUSION

- LE MODELE RELATIONNEL OFFRE UN CONCEPT DE VUE TRES PUISSANT
- METHODES D'INTERROGATION SIMPLES ET EFFICACES
- PROBLEMES EN SUSPENS :
  - MAJ AU TRAVERS DES VUES
  - VUES CONCRETES