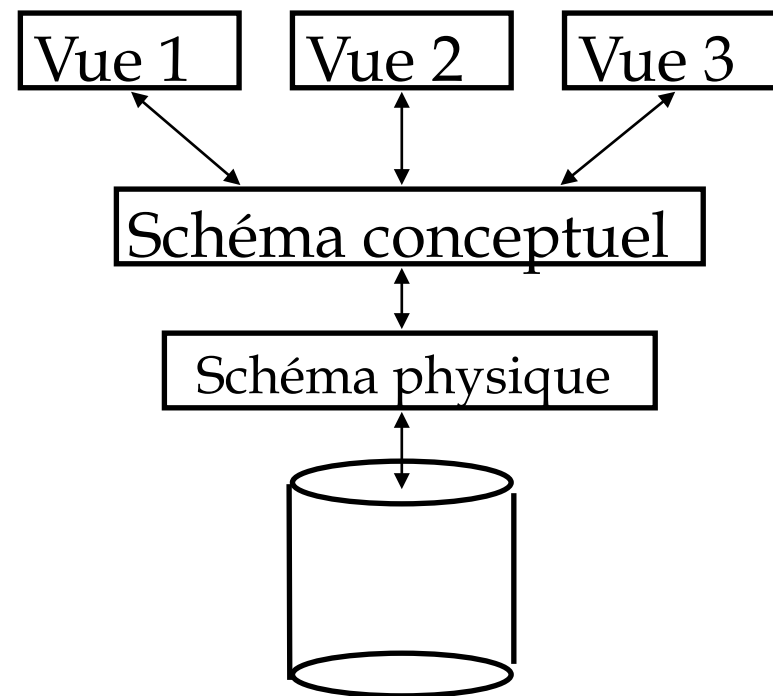


*Vues*

# Indépendance données / applications

Les 3 niveaux d'abstraction:

- Plusieurs vues, un seul schéma conceptuel (logique) et schéma physique.
  - Les vues décrivent comment certains utilisateurs/ groupes voient les données.
  - Le schéma conceptuel définit la structure logique des données.
  - Le schéma physique décrit les fichiers et les index utilisés.



# Définition

- Vue
  - Base de données virtuelle dont le schéma et le contenu sont calculés de la base réelle.
- Une vue est donc un ensemble de relations déduites d'une bases de données, par composition des relations de la base.
- Par abus de langage, une vue relationnelle est dite table virtuelle

# *Vues (externes)*

- **Objectif**
  - 1) Indépendance logique des applications par rapport à la base
  - 2) Vues pour la sécurité
- **Moyen**

Les vues sont des relations virtuelles dont la définition est stockée dans la méta-base.

Elles sont interrogées et mises à jour comme des relations normales
- **Avantages/Inconvénients**

Interrogation efficace

Mises à jour à travers les vues

# Création et Destruction

- Création

```
CREATE VIEW <NOM DE VUE> [ (LISTE D'ATTRIBUT)]  
AS <QUESTION>  
[WITH CHECK OPTION]
```

La clause WITH CHECK OPTION permet de spécifier que les tuples de la vue **insérés ou mis à jour** doivent satisfaire aux conditions de la question

- Destruction

```
DROP VIEW <nom de vue>
```

# Exemples

BUVEURS (NB, Nom, Prénom, Adresse, Type)

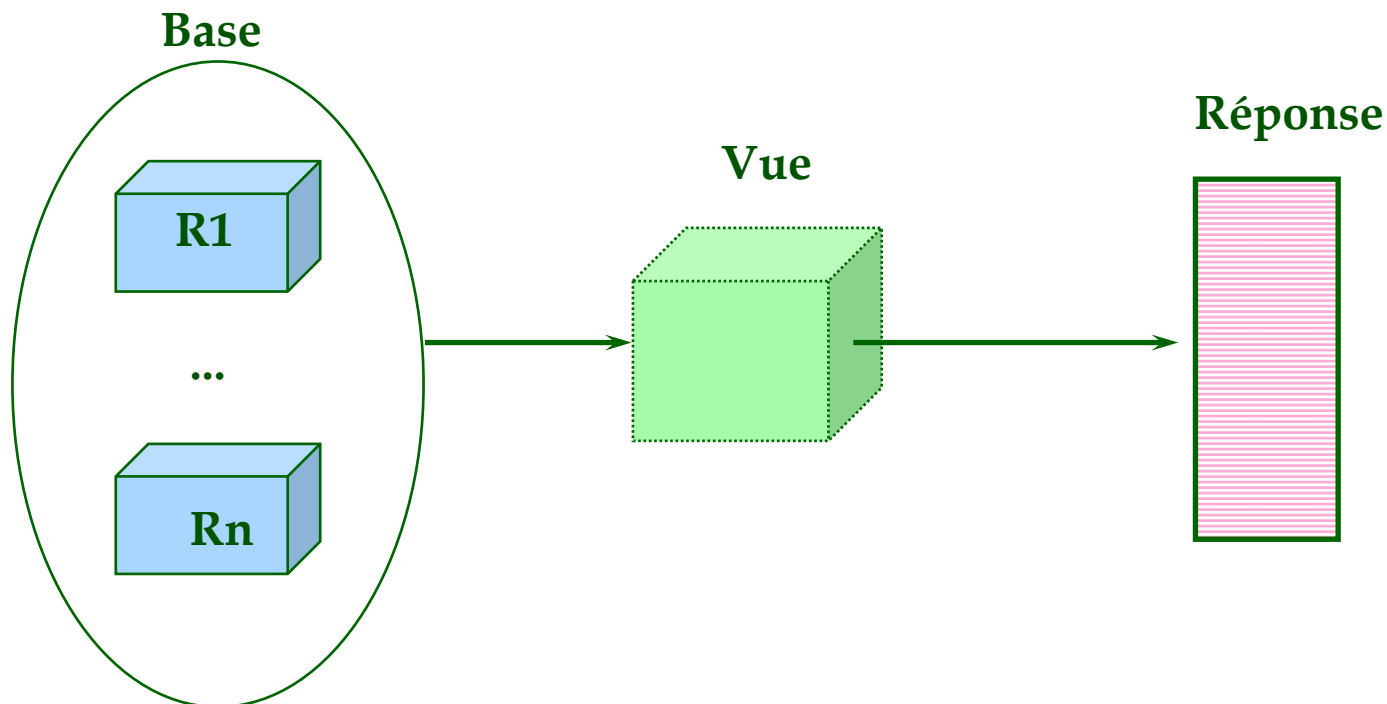
VINS (NV, Cru, Région, Millésime, Degré)

ABUS (NV, NB, Date, Quantité)

- (V1) Les vins de Bordeaux :  
CREATE VIEW vinsbordeaux (nv, cru, mill, degré) AS  
SELECT nv, cru, millésime, degré  
FROM vins WHERE Région = "Bordelais";
- (V2) Les gros buveurs :  
CREATE VIEW grosbuveurs AS  
SELECT nb, nom, prénom, adresse  
FROM buveurs b, abus a  
WHERE b.nb = a.nb and a.quantité > 10
- (V3) Les quantités de vins bues par cru :  
CREATE VIEW vinsbus (cru, mill, degré, total) AS  
SELECT cru, millésime, degré, SUM(quantité)  
FROM vins v, abus a WHERE v.nv = a.nv  
GROUP BY Cru

# *Interrogation de Vues*

- **Transparence pour l'utilisateur**  
manipulées comme des tables de la base
- **Simplifier les requêtes utilisateurs**
- **Utiles en architecture client-serveur**



# *Techniques d'évaluation*

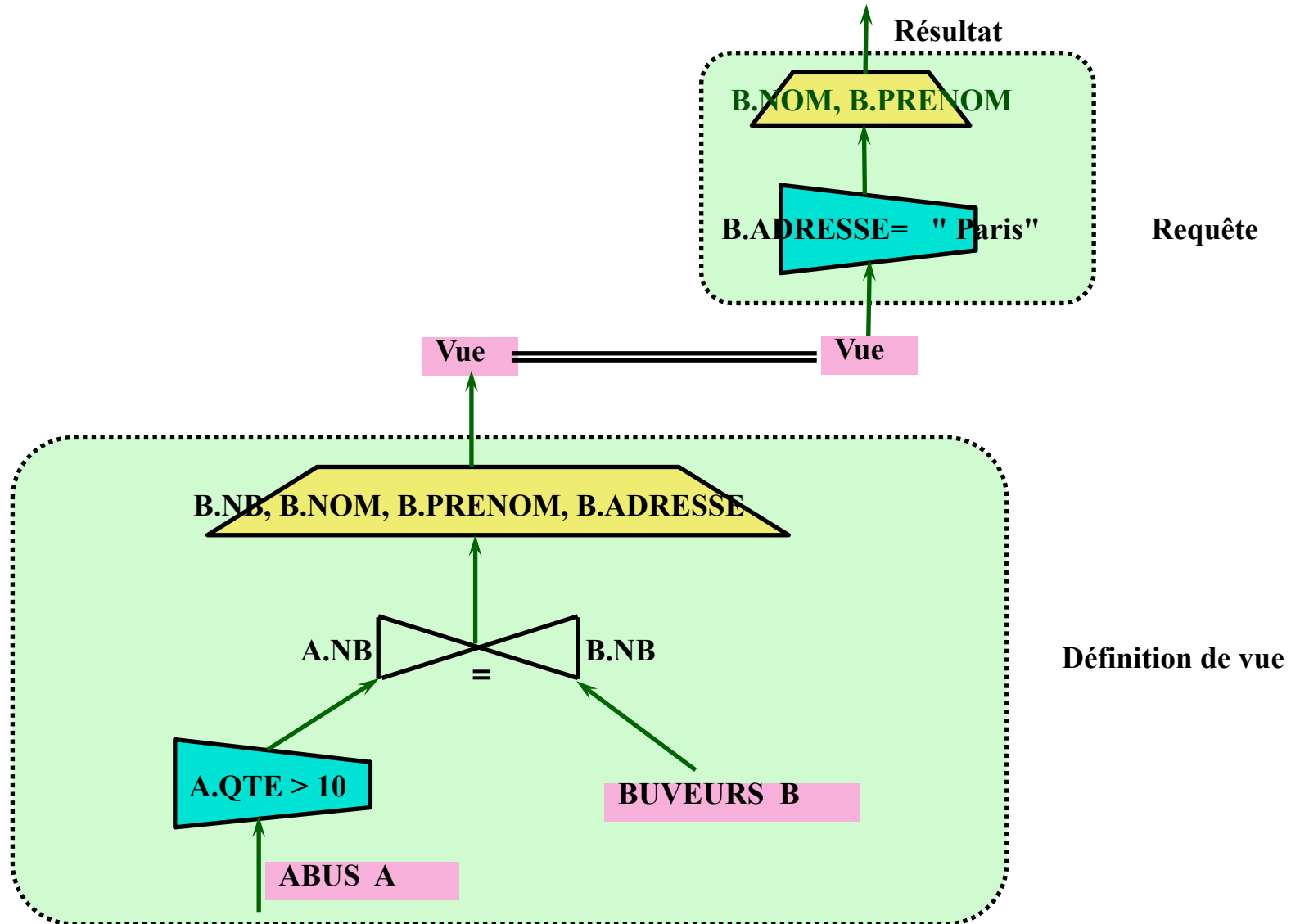
## Réécriture de requête

Peut être effectuée au niveau de l'expression SQL ou par concaténation d'arbres (plans d'exécution)

- Au niveau SQL, en remplaçant certaines relations virtuelles du FROM par leurs sources et en enrichissant les conditions de la clause WHERE pour obtenir le résultat de la question initiale.
- Concaténation d'arbre : mécanisme consistant à remplacer un nœud pendant dans un arbre relationnel par un autre arbre calculant le nœud remplacé.



# Concaténation d'arbres



# Exemple

- (1) Question

```
SELECT nom, prénom FROM grosbuveurs  
WHERE adresse LIKE "Paris".
```

- (2) Définition de vue

```
CREATE VIEW grosbuveurs AS  
SELECT nb, nom, prénom, adresse FROM buveurs b, abus a  
WHERE b.nb = a.nb AND a.quantité > 10.
```

- (3) Question modifiée

```
SELECT nom, prénom FROM buveurs b, abus a  
WHERE b.adresse LIKE " Paris" AND b.nb = a.nb AND  
a.quantité > 10.
```

# *Exemple*

```
SELECT nom, prenom FROM (SELECT nb, nom, prenom, adresse  
                           FROM buveurs b, abus a  
                           WHERE b.nb = a.nb AND a.quantite > 10)  
WHERE adresse LIKE "Paris" ;
```

VS.

```
SELECT nom, prenom FROM buveurs b, abus a  
WHERE b.adresse LIKE " Paris" AND b.nb = a.nb AND a.quantite > 10;
```

Sont elles équivalentes?

# *Éliminer l'imbrication du FROM (sémantique «bag» vs sémantique «set» )*

Set/set

```
SELECT DISTINCT a,b,c  
FROM (SELECT DISTINCT u,v  
      FROM R,S  
      WHERE ...), T  
WHERE ...
```

Devient

```
SELECT DISTINCT a,b,c  
FROM R, S, T  
WHERE ...
```

# *Éliminer l'imbrication du FROM (semantique «bag» vs semantique «set» )*

Set/Bag

```
SELECT DISTINCT a,b,c  
FROM (SELECT u,v  
      FROM R,S  
      WHERE ...), T  
WHERE ...
```

Devient

```
SELECT DISTINCT a,b,c  
FROM R, S, T  
WHERE ...
```

# *Éliminer l'imbrication du FROM (sémantique «bag» vs sémantique «set» )*

Bag/Bag

```
SELECT a,b,c  
FROM (SELECT u,v  
      FROM R,S  
      WHERE ...), T  
WHERE ...
```

Devient

```
SELECT a,b,c  
FROM R, S, T  
WHERE ...
```

# *Éliminer l'imbrication du FROM (sémantique «bag» vs sémantique «set» )*

Bag/Set

```
SELECT a,b,c  
FROM (SELECT DISTINCT u,v  
      FROM R,S  
      WHERE ...), T  
WHERE ...
```

Ne peut pas être transformé dans un seul select-from-where!

# *Partitionnement des données*

Parfois les données d'une base sont partitionnées:

- Horizontale: ex., projection sur certains attributs
  - utile quand certains attributs sont volumineux ou utilisés rarement
  - dans les bases de données reparties

Une vue avec **jointures** peut donner la vision d'ensemble

- Verticale: sélection sur certaines valeurs (ClientsParis, ClientsLyon)

Une vue avec **union** peut donner la vision d'ensemble



# *Vues pour la sécurité*

Compte(Nom, Type, Solde)

Bob n' a pas le droit d' accéder à l' attribut solde.

```
CREATE View CompteInfo AS  
SELECT nom, type  
FROM Compte;
```

Bob ne peut pas interroger Compte, peut interroger  
CompteInfo.

# *Mises à jour de vues*

- **Vue mettable à jour**

- Vue comportant suffisamment d'information pour permettre un report des mises à jour dans la base sans ambiguïté.

- **Conditions nécessaire**

- Clés des tables participantes déductibles de la vue

# *Exemple*

VINS (NV, Cru, Région, Millésime, Degré)

```
CREATE VIEW vinsbordeaux AS  
SELECT nv, millésime  
FROM vins WHERE Région = "Bordeaux"
```

```
INSERT INTO vinsbordeaux values ( '543' , 2001 )
```

```
INSERT INTO vins values( '543' , NULL, "Bordeaux", NULL, 2001)
```

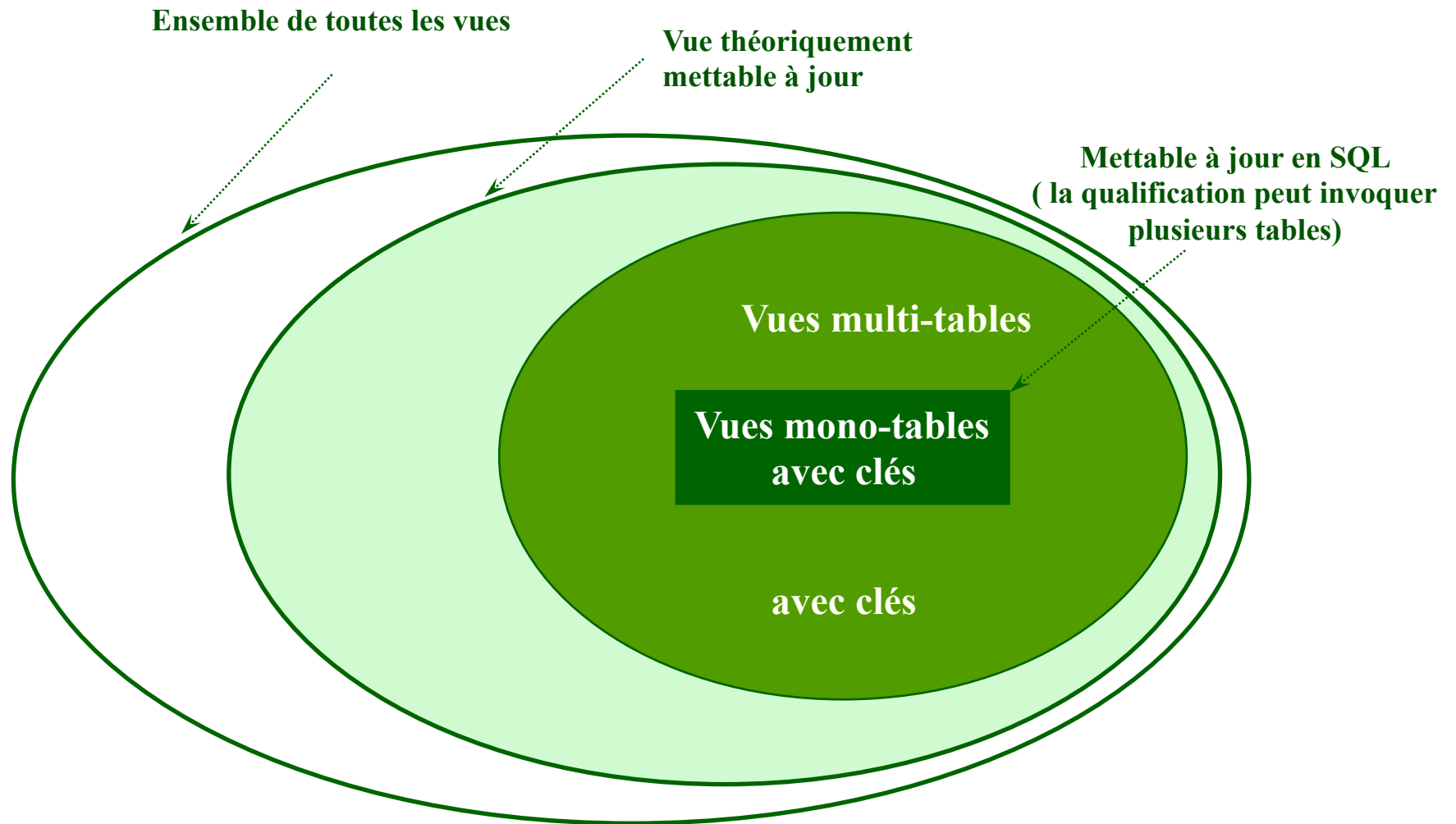
## *Exemple: Vue non-mettable à jour*

BUVEURS (NB, Nom, Prénom, Adresse, Type)  
VENTES (Acheteur, Vendeur, Produit)

```
CREATE VIEW Paris-vue AS  
SELECT vendeur, produit  
FROM buveurs, ventes  
WHERE buveurs.adresse = "Paris" AND buveurs.nom =  
ventes.acheteur;
```

```
INSERT INTO Paris-vue values ("Joe", "xyz") ?
```

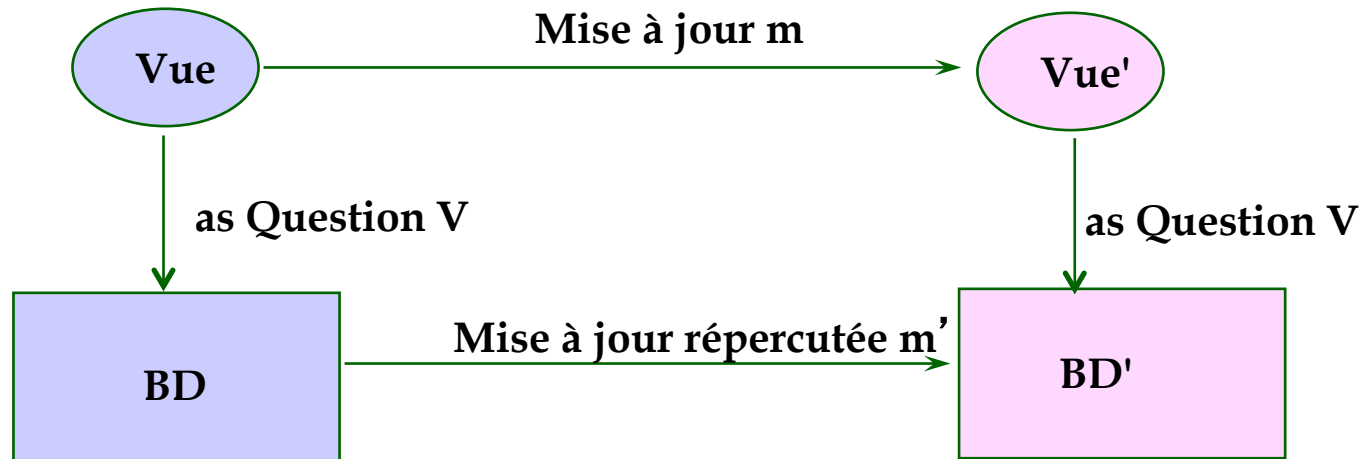
# Classification



# Principe

## ■ Problème

- Il peut manquer des données dans la vue pour effectuer le report dans la BD
- Comment définir une stratégie de report cohérente ?



- La mise à jour sur le contenu de la vue doit correspondre au contenu de la vue après le report de la mise à jour sur la BD.

## *With Check Option*

```
CREATE VIEW vinsbordeaux AS  
SELECT nv, région  
FROM vins WHERE Région = "Bordelais"
```

```
UPDATE vinsbordeaux SET région= "Alsace" ?
```

```
CREATE VIEW vinsbordeaux AS  
SELECT nv, région  
FROM vins WHERE Région = "Bordelais"  
WITH CHECK OPTION
```

# *Exemple de besoin de vues*

- **Besoin des entreprises**
  - accéder à toutes les données de l'entreprise
  - regrouper les informations disséminées dans les bases
  - analyser et prendre des décisions rapidement (OLAP)
- **Exemples d'applications concernées**
  - bancaire : regrouper les infos sur un client
    - réponse à ses demandes
    - mailing ciblés pour le marketing
  - grande distribution : regrouper les infos ventes
    - produits à succès
    - modes, habitudes d'achat
    - préférences par secteurs géographiques



# *Entrepôt de données*

- DataWarehouse

Ensemble de données historisées variant dans le temps, organisé par sujets, consolidé dans une base de données unique, géré dans un environnement de stockage particulier, aidant à la prise de décision dans l'entreprise.

- Trois fonctions essentiels :

- collecte de données de bases existantes et chargement
- gestion des données dans l'entrepôt
- analyse de données pour la prise de décision

- Vue matérialisée (concrète) :

- idéale pour modéliser un sous-ensemble de données extrait du datawarehouse et ciblé sur un sujet unique

# *Vue concrète*

- Vue concrète

**Table** calculée à partir des tables de la base par une question et matérialisée sur disques par le SGBD.

- Exemple : vues concrètes avec agrégats de la table  
VENTES(NUMV, NUMPRO, NUMFOU, DATE, QUANTITÉ, PRIX)

- Etendue selon les dimensions PRODUITS et  
FOURNISSEURS par les tables :

PRODUITS (NUMPRO, NOM, MARQUE, TYPE, PRIX)

FOURNISSEURS (NUMFOU, NOM, VILLE, RÉGION, TEL.)

# *Conclusion*

Deux types de vues, scénarios d' applications différents

Vues virtuelles

- Utilisées dans les bases de données
- Calculées « à la volé », lent à l' exécution (runtime)
- Toujours à jour

Vues matérialisées

- Utilisées dans les entrepôt de données
- Pre-calculées « offline », rapide à l' exécution
- Peuvent avoir des données « expirées »
- Les indexes sont des vues matérialisées