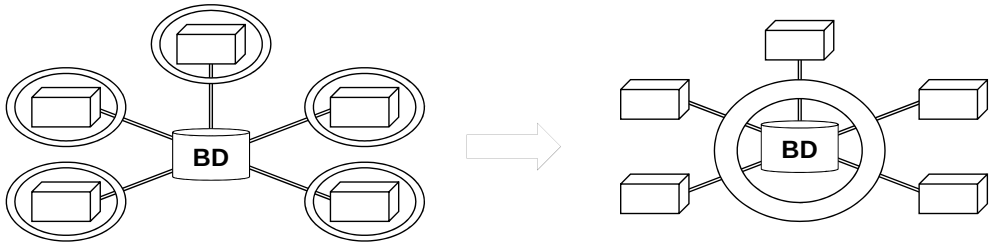
Intégrité des données

Définition des contraintes

Vérification des contraintes

4. Contraintes d'intégrité : Définition et objectif

- Contrainte d'intégrité :
  - propriété sémantique que doivent respecter les données afin d'assurer la cohérence de la base.
- Objectif : Détecter les mises à jour erronées et réagir
  - simplification du code des applications
  - sécurité renforcée par l'automatisation
  - évolutivité des contraintes
  - Cohérence globale des contraintes



Typologie des contraintes d'intégrité (1)

- Contraintes de domaine (mono-attribut)
  - Contrôle de types : ex: Nom alphabétique
  - Contrôle de valeurs : ex: Salaire mensuel entre 1 et 10 K€
  - Non Nullité : ex: le Nom d'un patient doit être renseigné
- Contraintes multi-attributs mono-tuple
  - Relations entre données élémentaires : PrixVente > PrixAchat
  - Relations temporelles : le salaire d'un employé ne peut pas décroître
- Contraintes multi-tuples mono-table
  - Unicité : le nro insee détermine un patient unique
  - Contrainte agrégative : salaire du PDG = max(salaire)

Typologie des contraintes d'intégrité (1)

- Contraintes multi-tuples multi-tables
  - Contrainte d'intégrité référentielle : une visite doit être liée à un médecin et un patient existants
  - Un électeur doit être inscrit sur au plus une liste électorale
  - Contrainte agrégative : la somme des quantités vendues doit être inférieure ou égale aux quantités produites
  - Le médicament X ne doit pas être prescrit en même temps que Y si une contre-indication est référencée dans le Vidal
- Problème complexe
  - Nécessite un langage de déclaration et un mécanisme de vérification
  - Les SGBD commerciaux supportent généralement peu de contraintes (par rapport à la norme SQL2)
    - Principalement CI de domaine, unicité, référentielle

Association des contraintes

- Une contrainte d'intégrité peut être :
  - Associée à un domaine
    - Spécifiée au travers de la clause CREATE DOMAIN
  - Associée à une table
    - Spécifiée au travers de la clause CREATE TABLE
  - Dissociées
    - Spécifiée au travers de la clause CREATE ASSERTION

Contraintes associées aux domaines

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

Exemple:

CREATE DOMAIN DATE_RDV IS DATE
DEFAULT (CURRENT_DATE)
CHECK (VALUE >= CURRENT_DATE)
NOT NULL

Tous les attributs variant sur ce domaine partageront ces même contraintes et valeur par défaut

Contraintes associées aux tables

create table <nom de table> (
<attribut> <domaine> **[<contrainte d'attribut>],** **(mono-attribut)**
<attribut> <domaine> **[<contrainte d'attribut>], …**
[<contrainte de table>]) **(mono ou multi-attributs)**

- Différent types de contraintes :
  - Non nullité : **not null**
  - Unicité : **unique**
  - Clé primaire : **primary key**
  - Prédicat quelconque : **check <formule>**
  - Contrainte d'intégrité référentielle : **reference**

Contraintes associées aux tables : syntaxe

< **def_contrainte_attribut** > ::= **[**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

- DEFERRABLE : vérification uniquement en fin de transaction plutôt qu'à chaque mise à jour
- Nommage des contraintes facilite la gestion des erreurs

Exemple récapitulatif

Visites ← **Prescriptions** → **Médicaments**

Id-V	Ligne	Id-M	Posologie
1	1	12	1 par jour
1	2	5	10 gouttes
2	1	8	2 par jour
2	2	12	1 par jour
2	3	3	2 gouttes
....	....	....	.....

Create Table Prescriptions(
Id-V integer,
Ligne integer,
Id-M integer,
Posologie varchar(200) **Not Null**,
Constraint NbMaxLigne **Check** (Ligne < 10),
Constraint Clé_Primaire_Presc **Primary Key** (Id-V, Ligne),

Aurait-on pu déclarer NbMaxLigne au niveau de l'attribut Ligne ?
Et Clé_Primaire_Presc au niveau de Id-V et Ligne ?

Contraintes référentielles

FOREIGN KEY (liste_colonnes**)**
REFERENCES nom_table **(**liste_colonnes**)**
[ON DELETE {RESTRICT | CASCADE | SET DEFAULT | SET NULL}]
[ON UPDATE {RESTRICT | CASCADE | SET DEFAULT | SET NULL}]
[NOT] 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**
- **Problème des contraintes référentielles croisées ==> mode DEFERRABLE**

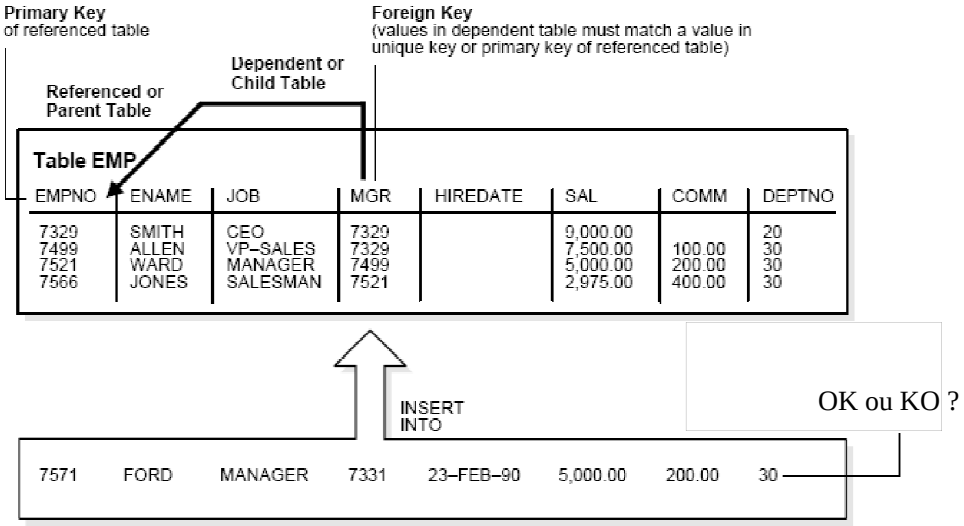
Exemple récapitulatif

Visites ← **Prescriptions** → **Médicaments**

Id-V	Ligne	Id-M	Posologie
1	1	12	1 par jour
1	2	5	10 gouttes
2	1	8	2 par jour
2	2	12	1 par jour
2	3	3	2 gouttes
....	....	....	.....

Create Table Prescriptions(
Id-V integer,
Ligne integer,
Id-M integer,
Posologie varchar(200) **Not Null**,
Constraint NbMaxLigne **Check** (Ligne < 10),
Constraint Clé_Primaire_Presc **Primary Key** (Id-V, Ligne),
Constraint Ref_Visites **Foreign Key** Id-V **References** Visites (Id-V)
on delete cascade,
Constraint Ref_Medic **Foreign Key** Id-M **References** Medicaments (Id-M)
on update cascade;

Contraintes référencielles intra-table



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** PRODUITS) >
(**SELECT** SUM(quantite) **FROM** LIVRAISONS) **)**

Triggers

Objectif
Syntaxe
Exemples

3. 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
  - Evénement = 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)

Déclencheurs : Objectifs

- Objectif : rendre la base de données '**active**'
  - maintenir des règles d'intégrité complexes
  - créer un audit de la base de données
  - dériver des données additionnelles (ex: statistiques)
  - mettre à jour des réplicas
  - déclencher des alertes
  - implanter des règles métier
- Gains (encore et toujours)
  - simplification du code des applications
  - sécurité renforcée par l'automatisation

Déclencheurs : Syntaxe

Create trigger <nom de trigger>
before | after — permet d 'indiquer quand le trigger va être exécuté

insert | delete | update [of <attributs>]	Quel est l'événement déclencheur
on <table>	indique le nom de la table qui doit être surveillée

E

[referencing old as <var>, new as <var>] — en SQL3, Oracle utilise :new et :old
for each row — Précise si l'action est exécutée 1 fois par tuple concerné ou pour toute la table

[when <condition>]	permet d 'indiquer une condition pour l'exécution d'un trigger ligne

C

DECLARE	Déclaration de variables pour le bloc PL/SQL
BEGIN	Bloc PL/SQL contenant le code de l'action à exécuter
<PL/SQL bloc>	Dans SLQ3, on peut indiquer une suite de commande SQL
END	

A

- La syntaxe et le type d'événements déclencheurs peuvent différer légèrement suivant le SGBD
  - Ex événements Oracle: DML et DDL statements, system events (startup, shutdown, errors), user events (logon, logoff)

Déclencheurs : Exemples simples

```
Create trigger calcul_TTC
after insert on Vente
For each row
Begin
    update vente set Prix_TTC = Prix_HT*1.206
End ;

Create trigger ModifCommande
after update on Commande
For each row
Begin
    if :new.qte< :old.qte
    then raise_application_error(-9996,' La quantité ne peut pas diminuer');
End ;

Create trigger ModifCommande
after update on Commande
For each row
When (new.qte< old.qte)
Begin
    raise_application_error(-9996, ,' La quantité ne peut pas diminuer');
End ;
```


Déclencheurs : un peu plus compliqué

REORDER Trigger

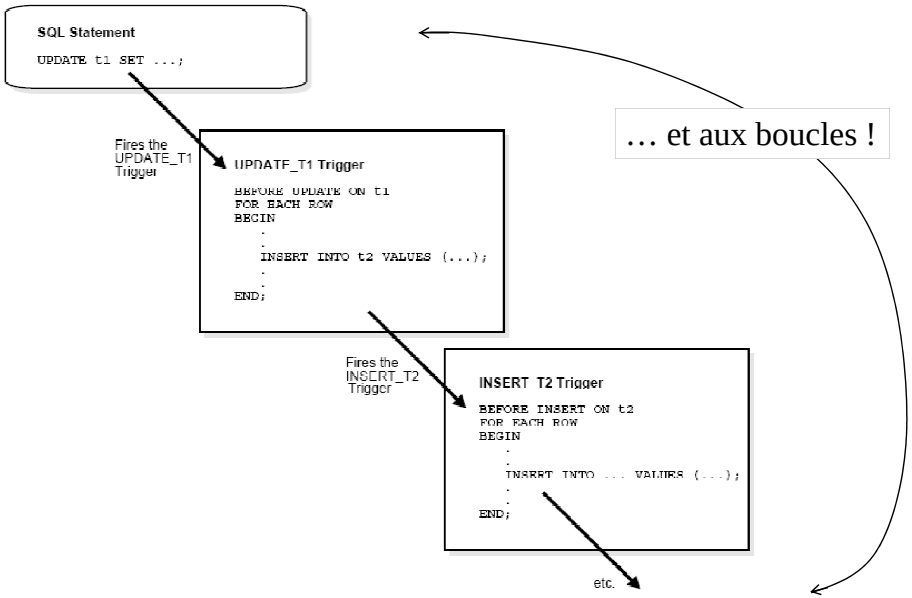
```
AFTER UPDATE OF parts_on_hand ON inventory          -- Triggering Statement

WHEN (new.parts_on_hand < new.reorder_point)       -- Trigger Restriction

                                                   -- Triggered Action
FOR EACH ROW
DECLARE                          /* a dummy variable for counting */
   NUMBER X;
BEGIN
   SELECT COUNT(*) INTO X        /* query to find out if part has already been */
   FROM pending_orders           /* reordered-if yes, x=1, if no, X=0 */
   WHERE part_no=:new.part_no;


IF X = 0
THEN                             /* part has not been reordered yet, so reorder */
   INSERT INTO pending_orders
   VALUES (new.part_no, new.reorder_quantity, sysdate);
 END IF;                         /* part has already been reordered */
END;
```


Déclencheurs : attention aux effets en cascade …



Vues externes

Définition des vues

Mécanisme d'interrogation

Vues concrètes

Les vues

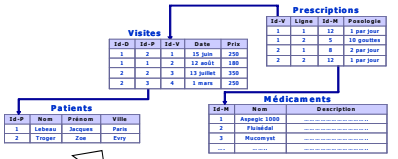
Les applications peuvent définir des **vues externes** de la BD

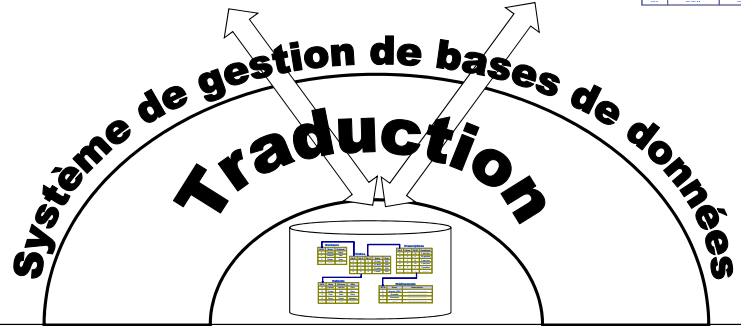
Gestion des médicaments

Nb_Presc_par_Médicament

Id-M	Nom	Description	Nombre
1	Aspegic 1000	.....	30
2	Fluisédal	.....	20
3	Mucomyst	.....	230
....	.....	.....	.....

Cabinet du Dr. Masse





Les vues : Définition et objectifs

- Définition
  - Une vue est une table virtuelle définie à partir d'autres tables
  - Sa définition est stockée dans la métabase
  - Son extension est calculée lors de l'interrogation
- Objectifs
  - **Evolution de la base de données** sans réécriture des applications : ajout / renommage de champs, de relation.
  - **Intégration d'applications existantes**
  - **Transparence à la localisation** (dans le cas de bases réparties)
  - **Confidentialité des données**
  - **Simplification** de l'écriture de requêtes
- Exemple

```
Create View Patients_Parisiens as (
        Select   Nom, Prénom
        From     Patients
        Where    Patients.Ville = 'Paris' )
```

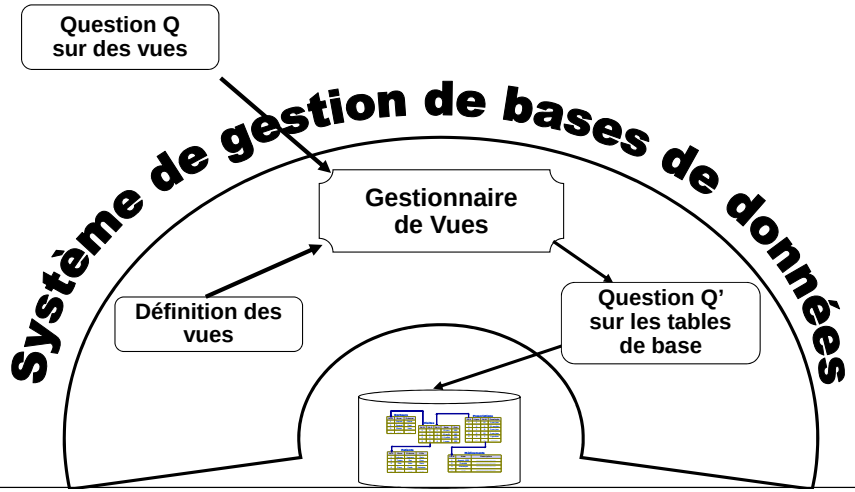

Syntaxe SQL

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

Les vues : mécanisme (1)

Le SGBD **transforme** la question sur les vues en question sur les tables de base

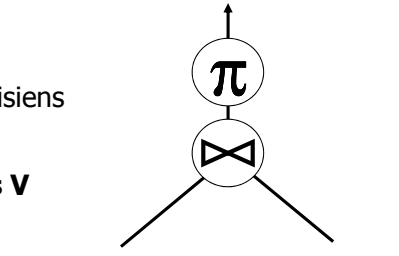


Les vues : mécanisme (2)

- Exemple

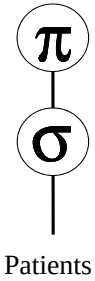
  Date des visites de tous les patients Parisiens

  Select Nom, Date
  From Patients_Parisiens P, Visites V
  Where P.Nom = V.Nom



Devient

Select Nom, Date
From Patients P, Visites V
Where Patients.Ville = 'Paris'
and P.Nom = V.Nom



Les vues : Mise à jour

- Non définie si la répercussion de la mise à jour vers la base de données est ambiguë
  - Comment répercuter la mise à jour d'un tuple de la vue calculant la moyenne des prix des médicaments ?
- Restrictions SQL (norme):
  - Pas de distinct, d'agrégats, ni d'expression de calcul
  - La vue contient les clés et les attributs « non nuls » des tables impliquées dans sa définition
  - Il y a une seule table dans le from !
  - Requêtes imbriquées possibles

  ➔ Certains SGBDs supportent plus de mises à jour

  ➔ possibilité de répercuter des mises à jour complexes via des triggers (triggers *instead of* dans Oracle)

Les vues matérialisées

- Vue matérialisée = cliché, instantané, snapshot, vue concrète
  - matérialisée sur disque
  - accessible seulement en lecture
  - peut être réactualisée
- Intérêt
  - Performance des requêtes complexes
  - Maintenance d'agrégats/résumés dans des entrepôts de données
  - Réplication de données sur un site distant ou sur un mobile
- Exemple
  - **create materialized view** Nb_Presc_par_Médicament **as**
    Select Id-M, Nom, Description, count(*)
    From Médicaments M, Prescriptions P
    Where M.Id-M = P.Id-M
    Group by M.id-M

    refresh every day

    ➔ Oracle: REFRESH [FAST | COMPLETE | FORCE] => différentielle, totale, automatique
    [ON COMMIT] | [ON DEMAND] |[START WITH date] [NEXT date]
    => périodicité: synchrone, asynchrone, cyclique

CONCLUSION

- Le modèle relationnel offre
  - des contraintes d'intégrité riches
  - Des mécanismes événementiels puissants
  - un concept de vues souple et simple

- Problèmes difficiles :
  - Contraintes d'intégrité avec agrégats
  - Triggers récursifs
  - maj au travers des vues