Intégrité des données

Définition des contraintes

Triggers

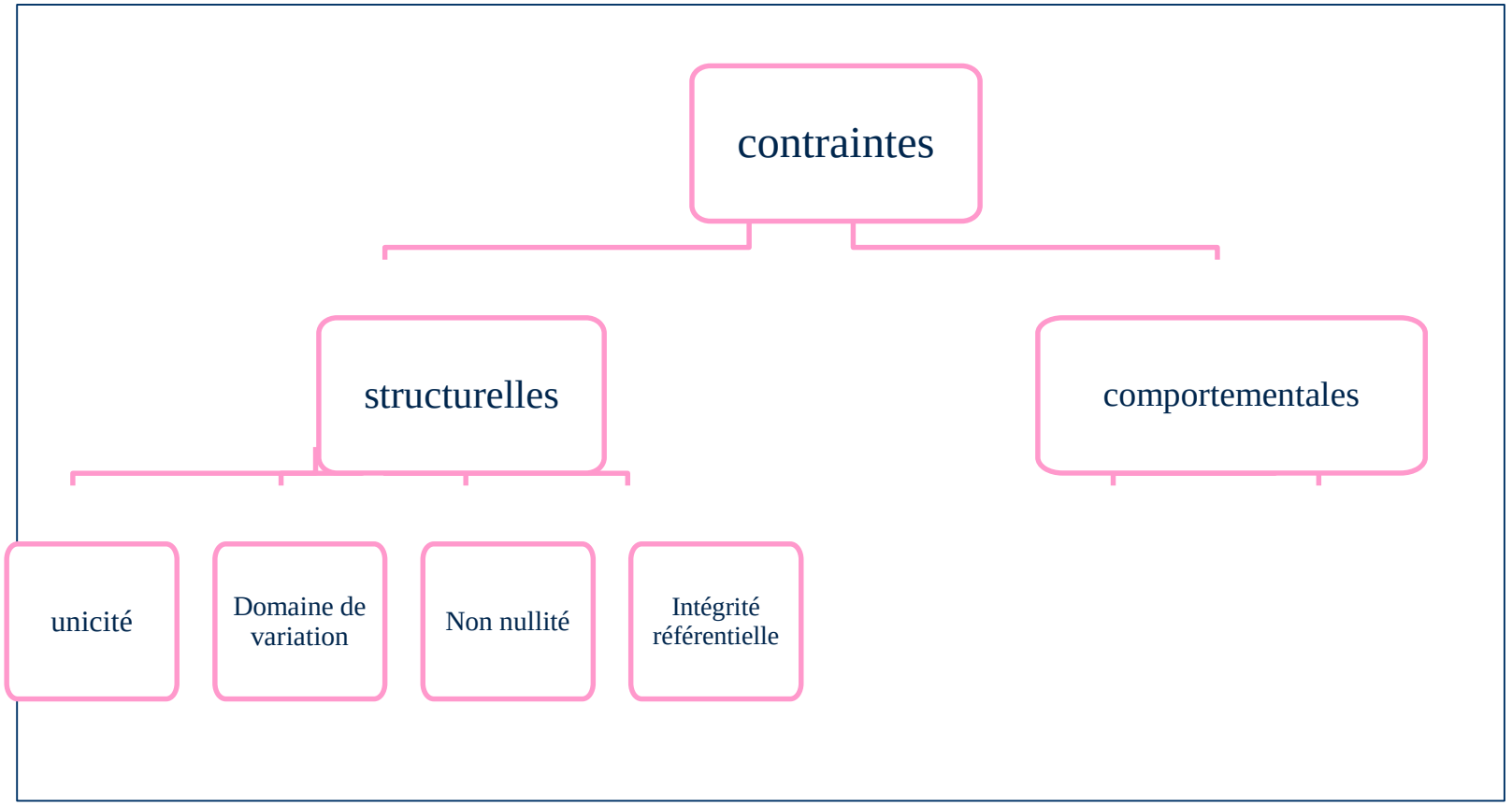
Partie 1:

CONTRAINTES

1. Définition des contraintes

- 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 et cohérence globale des contraintes

Typologie des contraintes



CONTRAINTES STRUCTURELLES

- Contraintes de DOMAINE
  - Permet de restreindre la plage de valeurs d'un domaine (liste de valeurs permises)
    - ex: le titre d'un ouvrage est de type chaîne de caractères

      Age compris entre 0 et100
- Contraintes d'ENTITE (unicité)
  - toute relation doit posséder au moins une clé et cette clé ne peut pas prendre de valeurs nulles
- Contrainte de non nullité

  La valeur d'un attribut doit être renseignée
- Contraintes REFERENTIELLE
  - ex: l'ensemble des valeurs de l'attribut employe.coddep doit être inclus dans l'ensemble des valeurs de l'attribut departement.coddep

Contraintes Comportementales

- CONTRAINTES COMPORTEMENTALES
  - Contraintes générales liées à une application spécifique
  - Elles concernent l'évolution des données lors des opérations de mise à jour.
  - Elles peuvent être exprimées sous forme d'assertions de la logique du 1er ordre, éventuellement temporelles.
  - ex: la somme des quantités consommées d'un produit doit être inférieure à la quantité produite de ce même produit.

Association des contraintes

- Une contrainte d'intégrité peut être :
  - 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> [DEFAULT valeur]
[**CONSTRAINT** nom_contrainte **CHECK (**condition**)**]

Exemple:

CREATE DOMAIN nom_filiere **CHAR(15) DEFAULT** ‘informatique'

CONSTRAINT filieres_possibles **CHECK**

(VALUE IN (‘informatique', ‘mathematique', ‘SNV'))

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 (exemple)

```
CREATE DOMAIN numero_dep  number DEFAULT  001
CONSTRAINT depots_possibles CHECK  (VALUE IN (001,002, 003,004))

Create table produit (numprod number primary key,
                      libelle varchar2(20),
                      prix number (6,2) check (prix between 0 and 100000));
Create table depot (numdep numero_dep primary key, adresse varchar2 unique);
Create table stock (numprod number references produit(numprod),
                    numdep number, qteS number not null,
                    constraint PK_stock Primary key (numprod, numdep),
                    constraint FK_stock foreign key (numdep) references
depot(numdep));
```

Contraintes référentielles

FOREIGN KEY (liste_colonnes**)**
REFERENCES nom_table **(**liste_colonnes**)**
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
[ON UPDATE {NO ACTION | 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**

La clause ON DELETE

- indique l'action que doit exécuter le système dans la table dépendante lors d'une suppression dans la table maître:
  - NO ACTION ne provoque aucune action,
  - CASCADE : il faut enlever les tuples correspondant de la table dépendante.
  - SET DEFAULT : il faut remplacer la clé étrangère des tuples correspondant de la table dépendante par la valeur par défaut qui doit être déclarée au niveau de l'attribut.
  - SET NULL a un effet identique, mais cette fois avec la valeur nulle.

La clause ON UPDATE

- indique comment le système doit modifier la clé étrangère dans la table dépendante lors de la mise à jour d'une clé primaire dans la table maître.
- Les effets sont identiques au ON DELETE, à ceci près que CASCADE provoque la modification de la clé étrangère de la même manière que la clé primaire.

Partie 2:

DÉCLENCHEURS

Déclencheurs **de Base de Données**

- Pour quoi faire ?
- Comment les
  - créer
  - activer
  - désactiver
  - Supprimer
- Exemples : application d'une règle de sécurité avec des déclencheurs

Déclencheurs BD - intérets

Stock

N°Pièce	quantitéDispo
100	12
105	23
110	50
220	200
222	201
228	87

J'ai besoin de toutes les pièces N° 110

Eh . . . mais il faudra remplir le stock de pièces N° 110 ! ! !

- En programmation classique :
  - écrire une routine ou une procédure appliquant la règle
  - appeler cette routine à chaque point précis de l'application où la règle doit être mise en oeuvre=> la règle n'est appliquée que si elle est invoquée explicitement
    => difficile d'obtenir la liste des endroits où la règle est mise en oeuvre

Déclencheurs - intérets

- Définition :
  Un déclencheur de base de données est un ensemble d'instructions qui s'exécutent lorsque le contenu d'une table est modifié par l'intermédiaire d'une instruction INSERT, UPDATE ou DELETE

- Les déclencheurs permettent :
  - d'implémenter des règles de sécurité
  - de contrôler la cohérence des données contenues dans une table lors de leur modification
  - d'effectuer des validations complexes en fonction de données provenant de tables multiples
  - d'implémenter des règles de gestion
  - de garder trace des modifications en reportant les valeurs modifiées dans une autre table
  - .....

Déclencheurs - intérêts

- Ces règles sont implémentées une fois pour toutes; elles n'ont pas à être vérifiées à chaque utilisation des tables sur laquelle elles portent
  - compatibilité avec le paradigme événementiel
    => pas d'invocation explicite de la règle
    => contrôle systématique de chaque modification des tables
  - facilité de mise à jour et de maintenance
    => code de la règle séparé de l'applicatif de traitement normal
    => contrôle centralisé de toutes les modifications d'une table

- Les déclencheurs sont des blocs PL/SQL qui s'exécutent *lorsqu' une table est modifiée* =>
  - impossible de savoir à l'avance quand un déclencheur sera exécuté
  - difficile de savoir quand un déclencheur s'exécute

Manipulation des déclencheurs BD

- créer
- activer
- désactiver
- supprimer

Création des déclencheurs

- Syntaxe :
  CREATE [OR REPLACE] TRIGGER <nom de déclencheur>
  {BEFORE | AFTER} <événement déclenchant> ON <nom table>
  [FOR EACH ROW]
  [WHEN (<condition>)]
  <Bloc PL/SQL>

  avec :
  <événement déclenchant> =
  - INSERT : insertion d'une nouvelle ligne dans la table
  - UPDATE : modifications des attributs d'une ligne de la table
  - DELETE : suppression d'une ligne de la table
  - une combinaison quelconque des trois (par ex : INSERT OR DELETE)

Création des déclencheurs - exemple

```
CREATE OR REPLACE TRIGGER maintientDuStock
AFTER INSERT OR UPDATE ON Stock
FOR EACH ROW
WHEN (:new.quantiteDispo < 10 OR :new.quantiteDispo IS NULL)
    BEGIN
        INSERT INTO Commandes (NProduit, quantiteCommandee)
            VALUES (:new.NProduit, 200) ;
    END ;
```

Déclencheurs de niveau instruction / ligne

- Un déclencheur de niveau instruction :
  - ***n'inclue pas la clause*** FOR EACH ROW dans l'instruction CREATE TRIGGER
  - ne s'exécute qu'une fois pour un événement particulier
  - ne peut accéder aux valeurs des lignes qui pourraient être modifiées par l'événement déclenchant
  - est adapté pour retrouver l'auteur ou la date de l'événement déclenchant

- Un déclencheur de niveau ligne :
  - ***doit inclure la clause*** FOR EACH ROW dans l'instruction CREATE TRIGGER
  - est déclenché pour chaque ligne modifiée par l'événement déclenchant
  - peut accéder aux anciennes et aux nouvelles valeurs modifiées par cet événement
  - est adapté pour mettre en application les règles de gestion et de sécurité

Référence aux colonnes de la table sur laquelle porte le déclencheur

- Dans le code associé aux déclencheurs de niveau ligne, on veut accéder aux valeurs des attributs de la ligne modifiée
  => utilisation de deux variables :
  - :old
  - :new

- Pour un déclencheur sur INSERT
  - les nouvelles valeurs sont dans :new.<nom d'attribut>
- Pour un déclencheur sur UPDATE
  - les anciennes valeurs sont dans :old.<nom d'attribut>
  - les nouvelles valeurs sont dans :new.<nom d'attribut>
- Pour un déclencheur sur DELETE
  - les anciennes valeurs sont dans :old.<nom d'attribut>

Déclencheurs BEFORE et AFTER

- Tout déclencheur BEFORE :
  - est exécuté avant que l'événement déclenchant n'ait lieu
- Tout déclencheur AFTER :
  - est exécuté après que l'événement déclenchant ait lieu

- Un déclencheur BEFORE de niveau ligne :
  - est exécuté avant que l'événement déclenchant n'ait lieu
  - peut affecter les valeurs de la ligne insérée ou modifiée
- Un déclencheur AFTER de niveau ligne :
  - est exécuté après que l'événement déclenchant ait lieu
  - ***ne peut pas*** affecter les valeurs de la ligne insérée ou modifiée

Récapitulatif des déclencheurs possibles

- Six déclencheurs de niveau ligne :
  - Before Insert, Before Upate, Before Delete
  - After Insert, After Update, After Delete
- Six déclencheurs de niveau instruction :
  - Before Insert, Before Upate, Before Delete
  - After Insert, After Update, After Delete

Activation / désactivation / suppression des déclencheurs

- Desactivation de déclencheur - Syntaxe :
  ALTER TRIGGER <nom du déclencheur> DISABLE ;

- Activation de déclencheur - Syntaxe :
  ALTER TRIGGER <nom du déclencheur> ENABLE ;

- Suppression de déclencheur - Syntaxe :
  DROP TRIGGER <nom du déclencheur> ;

Gérer les déclencheurs d'une table

- Désactiver ou réactiver tous les déclencheurs d'une table:

ALTER TABLE <nom-table> DISABLE | ENABLE ALL TRIGGERS

utiliser les prédicats conditionnels INSERTING/UPDATING/DELETING

- Les instructions:

  IF INSERTING THEN ……..

  ELSIF UPDATING (‘salaire’) THEN……..

  ELSIF DELETING THEN……..

- Insérées dans le corps du trigger, ces instructions permettent d’effectuer les traitements suivant les différentes actions.

Mise en œuvre d'une politique de sécurité avec des déclencheurs - exemple

- Domaine : crédit par carte
  - probabilité de fraude >80% lorsque le cumul des débits dépasse 5000 euros pour les trois derniers jours
  - décision : enregitrer tous les numéros de compte concernés dans une table séparée afin d'en faire une analyse détaillée

- Créer un déclencheur sur la table Debits
  - qui s'exécute après l'insertion d'une ligne dans cette table
  - qui évalue le montant des débits du compte concerné pour les trois derniers jours
  - qui insère une ligne dans la table ComptesMenacés si ce montant est supérieur à 5000

  Debits (numTransaction , numCompte, dateTransaction, montant , numCarte)
  ComptesMenacés (numCompte, numCarte, montantDerniersDebits, numTransaction)

Mise en œuvre d'une politique de sécurité avec des déclencheurs - exemple

```
CREATE OR REPLACE TRIGGER SurveillanceTentativesFraudes
AFTER INSERT ON  Debits
FOR EACH ROW
DECLARE
     montant3DerniersJours_v    number ;
BEGIN
SELECT SUM (montant)
INTO montant3DerniersJours_v
FROM Debits
WHERE  numCompte = :new.numCompte
              AND       dateTransaction <= SYSDATE -3 ;
. . .
```

Mise en œuvre d'une politique de sécurité avec des déclencheurs - exemple

```
IF (montant3DerniersJours_v > 5000) THEN
    INSERT INTO ComptesMenacés
        (numCompte, numCarte, montantDerniersDebits, numTransaction)
    VALUES
        (:new.numCompte, :new.numCarte,
        montant3DerniersJours_v, :new.numTransaction)
END IF;

END ;
```