



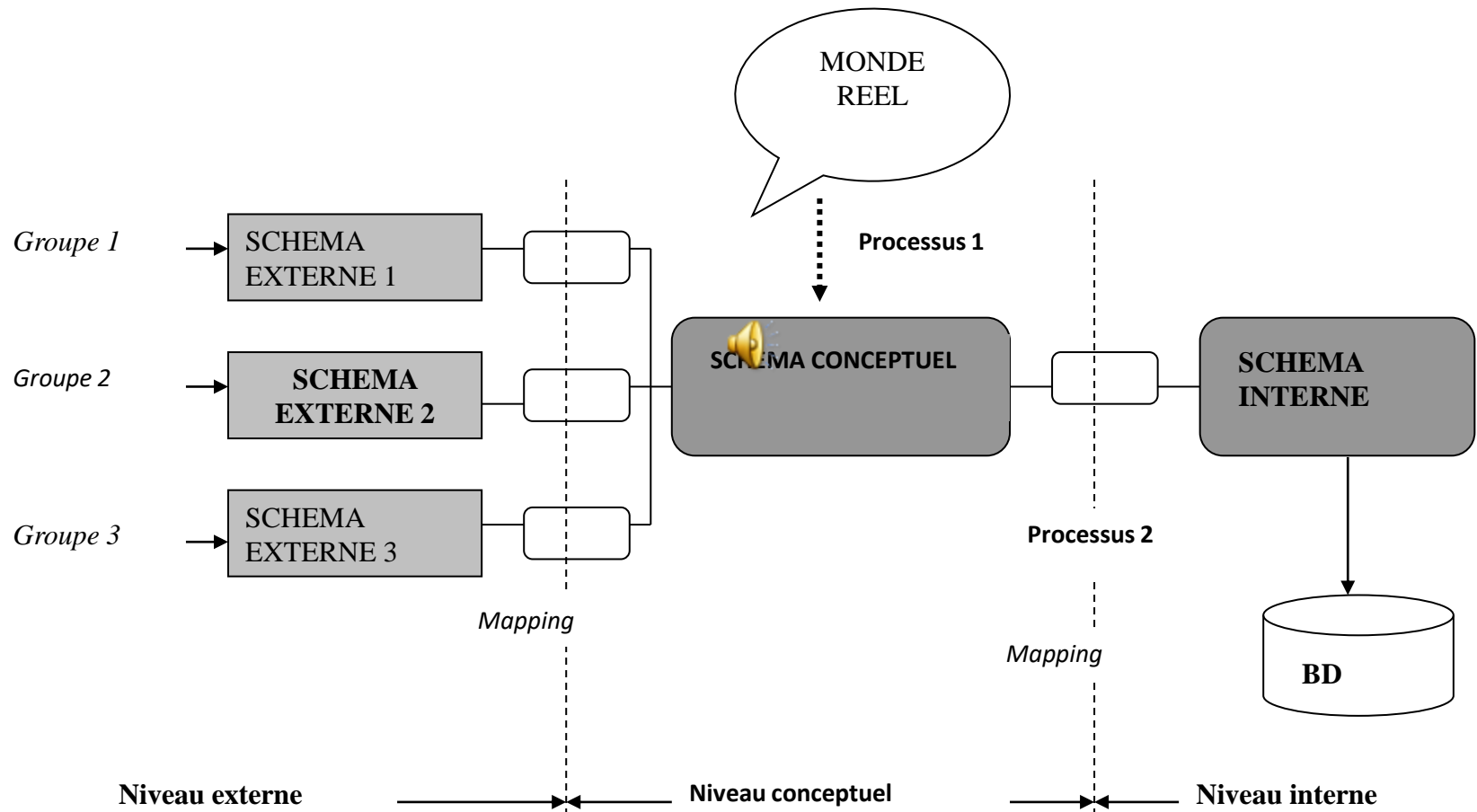
Architecture et fonctions d'un SGBD

Partie 3 : Vues externes &
Contraintes d'intégrité
Suite & Fin



Schémas Externes (Vues)

Niveaux de représentation des données



2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

La notion de vue va au-delà du simple sous-schéma comme sous-ensemble de schéma de la base.



Une vue est une relation virtuelle une fenêtre dynamique sur la base de données, c'est à dire qu'il n'existe pas de fichier qui représente directement une vue, par contre sa définition est stockée dans un catalogue appelé catalogue **Vue**.

2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes


La définition de la vue montre comment la vue est dérivée à partir des relations indiquées (relations sources). La commande permettant de définir une vue est :



```
CREATE VIEW <nom de vue> [(liste d'attributs)]  
AS <bloc de qualification> ;
```

2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

La définition de la vue montre comment la vue est dérivée à partir des relations indiquées (relations sources). La commande permettant de définir une vue est : 

CREATE VIEW <nom de vue> [(liste d'attributs)]
AS <bloc de qualification> ;

Le **bloc de qualification** commence par un SELECT et peut comporter toutes les possibilités que nous avons décrit dans une requête d'interrogation de SQL. Cependant, il faut souligner qu'aucun n-uplet n'est extrait de la base par une telle commande.

2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

La définition de la vue montre comment la vue est dérivée à partir des relations indiquées (relations sources). La commande permettant de définir une vue est :

```
CREATE VIEW <nom de vue> [(liste d'attributs)]  
AS <bloc de qualification> ;
```

Le **bloc de qualification** commence par un SELECT et peut comporter toutes les possibilités que nous avons décrit dans une requête d'interrogation de SQL. **Cependant, il faut souligner qu'aucun n-uplet n'est extrait de la base par une telle commande.**

2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

La commande permettant de définir une vue est :

```
CREATE VIEW <nom de vue> [(liste d'attributs)]  
AS <bloc de qualification> ;
```



Exemple :

```
CREATE VIEW ALGER-PIECE AS SELECT NP, NOM, POIDS  
FROM PIECE  
WHERE VILLE = 'ALGER';
```

La nouvelle relation ALGER-PIECE a trois attributs **hérités** de la relation PIECE. De nouveaux noms **auraient pu** être donnés aux attributs de cette nouvelle "relation".

2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

La commande permettant de définir une vue est: **CREATE VIEW** <nom de vue> [(liste d'attributs)]

AS <bloc de qualification> ;

Les noms des attributs **doivent** être spécifiés dans le cas d'une ambiguïté ou si le résultat du SELECT est une fonction d'agrégat.




Exemple :

```
CREATE VIEW PQ (NP, SOMQTE) AS SELECT NP, SUM( QTE )  
FROM FOURNITURE  
GROUP BY NP.
```

2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

Une vue constitue **une fenêtre dynamique** sur la base en ce sens que toute modification sur une relation source est automatiquement reportée au travers de la vue. Ainsi lorsqu'une pièce n'est plus stockée à Alger, elle n'apparaît plus dans la vue ALGER-PIECE.

Cependant mise à part  le fait **qu'il est impossible de définir des chemins d'accès dessus**, une vue se comporte comme tout autre relation de la base. En particulier on peut l'interroger et elle peut servir à la construction d'autres vues.

2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

La requête suivante permet de connaître les numéros des pièces stockées à Alger dont le poids est entre 20 et 40 :

```
SELECT NP  
FROM ALGER-PIECE  
WHERE POIDS BETWEEN 20 AND 40;
```



2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

```
SELECT NP  
FROM ALGER-PIECE  
WHERE POIDS BETWEEN 20 AND 40;
```

Pour traiter une telle requête, les SGBD relationnels effectuent ce qu'il est convenu d'appeler l'opération de composition de vue.

Il s'agit avant tout accès à la base de remplacer une vue par l'expression qui la définit jusqu'à former une requête qui ne porte que sur des relations de base. La requête précédente devient :

```
SELECT NP  
FROM PIECE  
WHERE POIDS BETWEEN 20 AND 40 AND VILLE = 'ALGER';
```



2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

Il y a une contre partie au fait que les vues soient des fenêtres dynamiques sur une base de données, autorisant ainsi différents usagers à avoir leur propre vision de tout ou partie de cette base : il n'est pas toujours possible d'effectuer des mises à jour au travers des vues.



2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes


Il y a une contre partie au fait que les vues soient des fenêtres dynamiques sur une base de données, autorisant ainsi différents usagers à avoir leur propre vision de tout ou partie de cette base : il n'est pas toujours possible d'effectuer des mises à jour au travers des vues.

En effet, pour faire **une mise à jour** au travers d'une vue, il doit être possible de **propager cette mise à jour sur les relations sources**. Si la vue est telle qu'il existe une correspondance biunivoque entre les n-uplets de la vue et ceux **d'une et une seule** relation source , **les mises à jours et les suppressions ne posent aucun problème et peuvent être propagées.**



2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

En ce qui concerne les **insertions**, si la vue est une **projection** d'une relation, il faut alors déterminer quelle sera la valeur des constituants n'apparaissant pas dans la vue. En particulier dans l'exemple **ALGER-PIECE**, l'insertion d'une nouvelle pièce au travers de la vue laisse indéfinie les valeurs **du matériau**. 

Au-delà de ces simples règles, la propagation des mises à jour au travers d'une vue quelconque est parfois dangereuse, voir impossible.

2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

Nous allons montrer sur un exemple que la mise à jour au travers d'une vue peut conduire à des incohérences. Considérons les deux relations :

EMP (NUM, NOM, NDEP*) 

DEPT (NDEP, DIR)

Qui ont pour signification respective qu'un employé a un numéro NUM, un nom NOM, qu'il travaille dans département de numéro NDEP et qu'un département a un numéro de département NDEP et un directeur DIR.

2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

EMP (NUM, NOM, NDEP*)

DEPT (NDEP, DIR)

L'extension de ces deux relations est :

EMP :

<u>NUM</u>	NOM	NDEP
E1	ALI	D1
E2	AMIN	D1
E3	SAMIR	D1
E4	SALIM	D2
E5	LYES	D2

DEP :

<u>NDEP</u>	<u>DIR</u>
D1	AMIN
D2	NADIR



Supposons une vue EMPDEP définie par la jointure de ces deux relations sur NDEP:

```
CREATE VIEW EMPDEP
AS SELECT NUM, NOM, EMP.NDEP, DIR
FROM EMP, DEP
WHERE EMP.NDEP = DEP.NDEP;
```

2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

EMP :

<u>NUM</u>	<u>NOM</u>	<u>NDEP</u>
E1	ALI	D1
E2	AMIN	D1
E3	SAMIR	D1
E4	SALIM	D2
E5	LYES	D2

DEP :

<u>NDEP</u>	<u>DIR</u>
D1	AMIN
D2	NADIR



```
CREATE VIEW EMPDEP
AS SELECT NUM, NOM, EMP.NDEP, DIR
FROM EMP, DEP
WHERE EMP.NDEP = DEP.NDEP;
```



<u>NUM</u>	<u>NOM</u>	<u>NDEP</u>	<u>DIR</u>
E1	ALI	D1	AMIN
E2	AMIN	D1	AMIN
E3	SAMIR	D1	AMIN
E4	SALIM	D2	NADIR
E5	LYES	D2	NADIR

2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

NUM	NOM	NDEP	DIR
E1	ALI	D1	AMIN
E2	AMIN	D1	AMIN
E3	SAMIR	D1	AMIN
E4	SALIM	D2	NADIR
E5	LYES	D2	NADIR

```
CREATE VIEW EMPDEP  
AS SELECT NUM, NOM, EMP.NDEP, DIR  
FROM EMP, DEP  
WHERE EMP.NDEP = DEP.NDEP
```



Si nous autorisons l'utilisateur à modifier la vue EMPDEP de la manière suivante :

```
UPDATE EMPDEP  
SET DIR = 'MADJID'  
WHERE NUM = 'E1';
```



En d'autres termes cela revient à dire que nous changeons le directeur de E1 qui de AMIN devient MADJID. En considérant le fait qu'un directeur est lié à un département de manière fonctionnelle, nous créons ainsi une incohérence car qui est le directeur du département D1 AMIN ou MADJID ? Un problème qui rejoint la conception de la base de données

2.2. Fonctions des SGBD relationnels

2.2. 4. Vues Externes

En outre, une mise à jour au travers d'une vue est impossible si, par exemple, l'expression qui la définit comporte une fonction de calcul.

Plusieurs études ont été consacrées au problème de mise à jour au travers des vues. Mais dans la pratique, les SGBD sont plus draconiens dans l'attitude vis à vis de ce problème en ce sens qu'une mise à jour au travers d'une vue V n'est autorisée que si :

- V est bâtie sur une seule relation source, disons R ,
- Il existe une association une à une entre les n -uplets de V et ceux de R de telle sorte qu'une opération de modification (I, S, M) sur V se traduit en Une opération sur R .

Cela signifie que les insertions, suppressions, et modifications ne sont possibles au travers d'une vue que lorsque celle-ci ne comporte pas :

- D'opération de jointure
- L'option GROUP BY ou UNIQUE
- De fonction de calcul





Contraintes d'intégrité

2.3 Problèmes de l'intégrité

A ce niveau nous allons décrire les mécanismes que doit fournir un SGBD pour assurer aux données qu'il gère un état cohérent et des possibilités de manipulations réservées aux usages dûment autorisés.

Nous distinguons plusieurs aspects :



- assurer la cohérence des informations stockées par rapport à leur signification : **contrainte d'intégrité** ou intégrité interne,
- Assurer l'accès concurrent tout en gardant la cohérence de la base : **synchronisation des accès concurrents**,
- Assurer la sûreté de fonctionnement en cas de panne : **mécanisme de reprise**,
- Assurer la sûreté d'utilisation : **droit d'accès**

2.3 Problèmes de l'intégrité

4.3.1. Les contraintes d'intégrité

Les données de la base doivent rester conformes à la réalité qu'elles représentent. Ceci est réalisé à l'aide de la définition des contraintes d'intégrité (CI). On distingue deux types de contraintes d'intégrité :



- Les contraintes d'intégrité statiques,
- Les contraintes d'intégrité dynamiques.

2.3 Problèmes de l'intégrité

4.3.1. Les contraintes d'intégrité

Les données de la base doivent rester conformes à la réalité qu'elles représentent. Ceci est réalisé à l'aide de la définition des contraintes d'intégrité (CI). On distingue deux types de contraintes d'intégrité :

- **Les contraintes d'intégrité statiques**, sont celles qui correspondent à un prédicat sur l'état courant de la base.
- Les contraintes d'intégrité dynamiques.



2.3 Problèmes de l'intégrité



4.3.1. Les contraintes d'intégrité

Les données de la base doivent rester conformes à la réalité qu'elles représentent. Ceci est réalisé à l'aide de la définition des contraintes d'intégrité (CI). On distingue deux types de contraintes d'intégrité :

- Les contraintes d'intégrité statiques,
- **Les contraintes d'intégrité dynamiques:** sont celles qui concernent le passage d'un état à un autre.

2.3 Problèmes de l'intégrité

4.3.1. Les contraintes d'intégrité

A tout instant de l'existence d'une base de données, on doit pouvoir définir une **nouvelle contrainte d'intégrité**. Le **SGBD** doit fournir des mécanismes pour vérifier que la base est cohérente vis à vis de cette contrainte et **l'accepter ou la rejeter**. Lorsqu'elle est acceptée, une contrainte doit être conservée dans un **catalogue** de la base. Nous allons énumérer les différents types de contraintes qu'on rencontre généralement :



2.3 Problèmes de l'intégrité

4.3.1. Les contraintes d'intégrité



A tout instant de l'existence d'une base de données, on doit pouvoir définir une nouvelle contrainte d'intégrité. Le SGBD doit fournir des mécanismes pour vérifier que la base est cohérente vis à vis de cette contrainte et l'accepter ou la rejeter. Lorsqu'elle est acceptée, une contrainte doit être conservée dans un catalogue de la base. Nous allons énumérer les différents types de contraintes qu'on rencontre généralement :

- **Unicité de la clé** d'une relation. Dans le langage SQL, cela exprimable grâce à la commande **CREATE UNIQUE INDEX** : où à chaque création d'un n-uplet il faut s'assurer que la valeur de la clé n'existe pas déjà dans la base.

2.3 Problèmes de l'intégrité

4.3.1. Les contraintes d'intégrité

Nous allons énumérer les différents types de contraintes qu'on rencontre généralement :



- **Unicité de la clé** d'une relation.
- **Contrainte d'intégrité individuelle** ou contrainte que doit vérifier un n-uplet individuellement. Il en existe de plusieurs sortes :
 - **plage de valeur**, par exemple, $4.000 \leq \text{salaire} \leq 20.000$
 - **liste** de couleurs possibles [bleu, rouge, vert, jaune]
 - contraintes **entre constituants** : $\text{QTE-STOCK} \geq \text{QTE-COMMANDE}$
 - contraintes de **format** : $\text{NOM CHAR}(20)$

2.3 Problèmes de l'intégrité

4.3.1. Les contraintes d'intégrité

Ces contraintes s'adressent au n-uplet, il est donc facile de les tester lorsqu'on met à jour ce n-uplet. Dans le langage SQL, on dispose d'une instruction **ASSERT** pour exprimer une contrainte d'intégrité sur une relation :



ASSERT <nom assertion> [on relation] : prédicat ;

Par exemple, pour assurer que le matériau des pièces ne peut prendre que quatre valeurs on écrit :

```
ASSERT CI1 ON PIECE : MATERIAU IN ('fer', 'bronze', 'zinc',  
                                     'plomb') ;
```

Toute **requête de modification est transformée** avant son exécution pour y faire apparaître la contrainte comme une condition sur la relation.

2.3 Problèmes de l'intégrité

Contrainte intra-relation :



cas où l'on veuille contrôler la valeur d'un constituant d'un n-uplet en fonction des valeurs de ce constituant pour les autres n-uplets.

On dira alors que l'on a **une contrainte intra relation verticale**. De la même manière, on pourra contrôler la valeur d'un constituant en fonction des valeurs apparaissant dans les autres constituants. On dira alors qu'on a **une contrainte intra-relation horizontale**.

2.3 Problèmes de l'intégrité



Contrainte intra-relation :

Exemple.

Soit la relation décrivant les employés d'un grand magasin :

Employé(Nom, Salaire, Nom-Rayon)

On veut exprimer la contrainte d'intégrité suivante : " Un employé ne peut gagner plus du double de la moyenne des salaires de son rayon"

En SQL on écrira : `ASSERT CI2 ON Employé E : Salaire ≤
2 * (SELECT AVG (Salaire)
FROM Employé
WHERE Nom-Rayon = E.Nom-Rayon);`

2.3 Problèmes de l'intégrité

Contrainte intra-relation :

Exemple.

Soit la relation décrivant les employés d'un grand magasin :

Employé(Nom, Salaire, Nom-Rayon)

ASSERT C12 ON Employé E : Salaire \leq

2 * (SELECT AVG (Salaire)

FROM Employé

WHERE Nom-Rayon = E.Nom-Rayon);



Ce genre de contrainte **s'avère déjà plus difficile à mettre en œuvre** au niveau du SGBD. Lors de toute modification sur les relations, il faut effectuer un calcul qui peut s'avérer **long et coûteux**.

2.3 Problèmes de l'intégrité

Contrainte intra-relation :

Exemple.

Considérant une autre relation décrivant le magasin :

Rayon (Nom-Rayon, Etage, Recettes, Dépenses, Nb-Employés)



Pour exprimer que les dépenses doivent être inférieures ou égales aux recettes, on écrira :

ASSERT CI3 On Rayon : Dépenses \leq Recettes

2.3 Problèmes de l'intégrité

Contraintes sur l'ensemble de n-uplets :

Avec la relation Employé, on veut exprimer la contrainte : "la moyenne des salaires des employés du rayon aliment doit être inférieur à 10.000".

[illegible]

2.3 Problèmes de l'intégrité

Contraintes inter-relations :

Dans une base de données les relations sont généralement liées les unes aux autres. Il faut donc pouvoir exprimer les contraintes entre des relations. On peut vouloir ainsi contrôler que l'ensemble des valeurs d'un constituant d'une relation R1 apparaît dans également dans une relation R2. Exprimer le fait que tout rayon apparaissant dans la relation Employé est décrit dans la relation Rayon, en SQL s'écrit :



```
ASSERT CI5 (SELECT NOM-Rayon FROM Employé)  
           IS IN (SELECT Nom-Rayon  
                  FROM Rayon);
```

2.3 Problèmes de l'intégrité

A l'opposé des contraintes d'intégrité statiques que nous venons de voir on considère des contraintes sur le passage d'un état à un autre. En d'autres termes, les n-uplets du nouvel état dépendant de ceux de l'ancien. Pour exprimer **une contrainte dynamique** sur les valeurs, il faut pouvoir dans le langage relationnel faire une référence explicite à l'ancienne valeur (OLD) et à la valeur Nouvelle (New). Si on veut exprimer le fait que le salaire d'un employé ne peut diminuer, on écrit en SQL :

ASSERT C16 **ON UPDATE** OF Employé (Salaire):



NEW Salaire \geq OLD Salaire;

2.3 Problèmes de l'intégrité

A l'opposé des contraintes d'intégrité statiques que nous venons de voir on considère des contraintes sur le passage d'un état à un autre. En d'autres termes, les n-uplets du nouvel état dépendant de ceux de l'ancien. Pour exprimer une contrainte dynamique sur les valeurs, il faut pouvoir dans le langage relationnel faire une référence explicite à l'ancienne valeur (OLD) et à la valeur Nouvelle (New). Si on veut exprimer le fait que le salaire d'un employé ne peut diminuer, on écrit en SQL :

ASSERT C16 **ON UPDATE** OF Employé (Salaire)



NEW Salaire \geq OLD Salaire;

Remarquons ici que l'on indique le moment où cette contrainte doit être vérifiée, à savoir lors d'une mise à jour.

2.3 Problèmes de l'intégrité

Un autre **aspect des contraintes d'intégrité dynamiques** est celui qui consiste à définir des actions spontanées qui seront **déclenchées automatiquement** par le SGBD lorsque certains éléments se produiront.

Exemple

On veut exprimer que lorsqu'un nouvel employé est inséré dans la base, le nombre d'employés figurant dans la relation rayon doit augmenter de un pour le rayon du nouvel employé. Dans SQL, on définit alors une action spontanée ou **TRIGGER** de la manière suivante :

```
Create or replace TRIGGER EMPINS
    AFTER INSERT ON Employé
    (UPDATE RAYON
     SET Nb-Employés = Nb-Employés + 1
     WHERE Nom-Rayon = NEW Employé.Nom-Rayon);
```



2.3 Problèmes de l'intégrité

Pour que la base soit toujours cohérente, il faut également décrire ce qui doit se passer lors d'une suppression d'un employé ou lorsqu'une mise à jour est effectuée sur un employé. On aura donc :

Create Or Replace Trigger EMPSUP



```
AFTER DELETE ON Employé (UPDATE RAYON
SET Nb-Employés = Nb-Employés – 1
WHERE Nom-Rayon = OLD Employé.Nom-Rayon);
```

Create Or Replace TRIGGER EMPMAJ

```
AFTER UPDATE ON Employé (UPDATE RAYON
SET Nb-Employés = Nb-Employés – 1
WHERE Nom-Rayon = OLD Employé.Nom-Rayon
UPDATE RAYON SET Nb-Employé = Nb-Employé + 1
WHERE Nom-Rayon = NEW Employé.Nom-Rayon);
```

2.3 Problèmes de l'intégrité

Contraintes différées ou immédiates : est une dernière façon de caractériser les contraintes d'intégrité. Elle concerne le moment où elles doivent être vérifiées par le SGBD. Le qualificatif "**immédiate**" signifie que **dès sa définition** une contrainte d'intégrité sera **vérifiée** pour savoir si la base est cohérente. Par contre, une contrainte d'intégrité peut être **différée** ce qui signifie que le test de cohérence de la base sera fait à un **autre moment** qu'à la **définition**. Cette notion est liée au concept de transaction. Une transaction est une unité atomique d'exécution.

Elle doit effectuer plusieurs modifications dans la base et ou bien toutes sont réalisées ou bien aucune ne l'est. Pendant l'exécution d'une transaction la base peut se trouver temporairement dans un état incohérent et la vérification des contraintes d'intégrité sera différée après la fin de la transaction.

2.3 Problèmes de l'intégrité

Notons que pour un problème de performance, les SGBDs commercialisés ne vérifient pas tous ces types de contraintes. Notons également que la notion de dépendance fonctionnelle (entre constituant d'une relation) au niveau de la conception d'une base de données est un autre type de contrainte d'intégrité.

