Contraintes et Triggers en SQL

Objectif :

Définir et créer des éléments actifs permettant de garantir l'intégrité des données.

Contrainte d'intégrité :

- Dans le cas général : formule de la logique du premier ordre.
- En pratique : clefs, contraintes référentielles, contraintes de domaine, contraintes globales.
- Distinction car différents outils associés.

Elément actif :

Expression (ou statement) écrit une fois, stocké dans la base qui réagit de façon opportune.

Principales Contraintes

- Clefs

$$\forall\ t_1,\ t_2 \in R(K,U\text{-}K) :$$
$$K(t_1) = K(t_2) \Rightarrow U\text{-}K(t_1) = U\text{-}K(t_2)$$

où K est une clef pour R

- Intégrité réferentielle

$$\forall\ t_1 \in R_1(K_1,\ K_2,\ U_1),\ \forall\ c \in Dom(K_2),\ \exists\ t_2 \in R_2(K_1,\ K_2,\ U_2) :$$
$$K_2(t_1) = c \Rightarrow K_2(t_2) = c$$

où K_1 est une clef pour R_1 et K_2 une clef pour R_2 et où c est une constante

Principales Contraintes

- Contraintes de domaine.

Restreignent le domaine d'un attribut.

$$\forall\ t \in R(X, U\text{-}X) :$$
$$X(t) = c \Rightarrow c \in Dom(X) \cap Val$$

où c est une constante et Val un ensemble strictement inclus dans Dom(X)

- Contraintes globales.

Sont exprimables par une formule du premier ordre.

$$\forall\ p \in Personnes(nom,\ age),\ \forall\ v \in Vehicule(marque,\ proprietaire) :$$
$$(proprietaire(v) = nom(p) \wedge marque(v) = \text{“Ferrari”}) \Rightarrow age(p) \geq 40$$

Problème

- Mises à jour potentiellement incorrectes
- Outils pour en assurer la correction.
- Certains ont déjà été vus ... (Si, si :-))

Analogie GL : les contraintes sont des invariants que le SGBD doit préserver

Principaux Outils

Niveau schéma

- Clefs
- Clefs étrangères
- Assertions
- Triggers (Déclencheurs).

Clefs

- Déclaration

```
1) CREATE TABLE Star(
2)        nom           CHAR(30) PRIMARY KEY,
3)        adresse       VARCHAR(255),
4)        sexe          CHAR(1),
5)        datenaiss     DATE
          );

1) CREATE TABLE Star(
2)        nom           CHAR(30),
3)        adresse       VARCHAR(255),
4)        sexe          CHAR(1),
5)        datenaiss     DATE,
6)        PRIMARY KEY   (nom)
          );
```

Clefs

- Variantes

```
2) nom CHAR(30) UNIQUE
```

Ou bien :

```
6) UNIQUE(nom)
```

- Mais aussi

```
2) nom      CHAR(30)     UNIQUE,
3) adresse VARCHAR(255) UNIQUE,
```

Primary Key ou Unique ?

Une déclaration de clef primaire est presque un synonyme de unique. Les différences sont :

1. *Une seule clef primaire* alors que plusieurs attributs peuvent être déclarés unique.

2. Une clef étrangère *ne peut référencer qu'une clef primaire.*

3. Le constructeur de SGBD (Oracle par ex.) a l'option d'attacher au concept de clef primaire, une sémantique particulière (ne faisant pas partie du standard SQL2). Par exemple, *automatiquement associer un index à une clef primaire* et demander explicitement au programmeur d'application de demander une création d'index pour les attributs uniques. Ou encore *automatiquement trier* la relation selon les attributs constituant la clef primaire.

Gérer les Contraintes de Clefs

- Le système le fait pour vous pour les mises à jour significatives (inserts et updates) *mais*

- L'utilisation et donc la déclaration d'index a bon escient peut améliorer les performances !

```
CREATE UNIQUE INDEX IndexAdresse ON Star(adresse)
```

ou même si c'est absurde dans ce cas !

```
CREATE INDEX IndexClef ON Star(nom, adresse)
```

Intégrité Référentielle et Clefs Etrangères

Situation suivante :

Relation `Studio` attribut `Idpres` (identificateur du président du studio).

Comment exprimer le fait que tout président de studio est aussi producteur (que `Idpres` correspond bien à une valeur pour `Idprod` dans la relation `Producteurs`.

```
CREATE TABLE Studio(
             nom CHAR(30) PRIMARY KEY,
             adresse VARCHAR(255),
             Idpres INT REFERENCES Producteur(Idprod)
             );
```

Attention : `Idprod` doit être PRIMARY KEY dans `Producteur`

Gestion de l'Intégrité Réferentielle

Les trois types de mises à jour sont concernés : insert, update et delete.

Trois politiques de gestion sont alors disponibles :

1. La politique par défaut : rejet des modifications incorrectes.

2. La politique en cascade et

3. La politique Set-Null.

La politique par défaut

Suivant le type de mise à jour :

1. `INSERT` d'un nouvel n-uplet dans `Studio` dont la valeur de l'attribut `presId` est `NULL` → Rejet par le SGBD.
2. `UPDATE` de l'attribut `Idpres` d'un n-uplet dans `Studio` à une valeur non NULL mais apparaissant pas dans `Producteur` → Rejet par le SGBD.
3. `DELETE` d'un n-uplet dans `Producteur` dont l'attribut `prodId` est référencé par un n-uplet de `Studio` → Rejet par le SGBD.
4. `UPDATE` de l'attribut `prodId` d'un n-uplet de `Producteur` référencé par un n-uplet de `Studio` → Rejet par le SGBD.

Les Politiques en Cascade et Set-Null

Cascade Avec ce type de politique, les cas 3 et 4 du transparent précédent sont gérés de la façon suivante :

- Cas 3 : destruction des n-uplets référençant la valeur détruite dans `Studio`.
- Cas 4 : Propagation de la mise à jour dans `Studio`

Set-Null

Affecte, dans la relation `Studio`, la valeur `NULL` à l'attribut `Idpres` pour tous les n-uplets concernés. (cas 3 et 4).

Exemple

```
CREATE TABLE Studio(
        nom CHAR(30) PRIMARY KEY,
        adresse VARCHAR(255),
        Idpres INT REFERENCES Producteur(Idprod)
               ON DELETE SET NULL
               ON UPDATE CASCADE
      );
```

Contraintes de Domaine

NOT NULL

Déjà vu.

Attention avec la politique Set-Null

Vérification sur valeurs d'attributs

```
4)      Idpres INT REFERENCES Producteur(Idprod)
                                 CHECK (Idpres >= 100000)
```

Ou encore

```
4)      sexe CHAR(1) CHECK (sexe IN ('F', ''M'))
```

Vérification sur valeurs d'attributs

- La vérification est opérée lorsqu'un nouvel n-uplet est inséré ou lorsqu'il est modifié. Si la contrainte est violée, la mise à jour est rejetée par le SGBD.

- Attention

```
4)  Idpres INT CHECK
          (Idpres IN (SELECT Idprod FROM Producteur))
```

- Cette expression est licite

Vérification sur valeurs d'attributs

- Si insertion ou update de Idpres avec une valeur qui ne correspond à aucun identificateur de producteur, la mise à jour est rejetée. Ce qui est souhaité. mais

- Si une destruction dans Producteur a pour effet de faire disparaitre une valeur référencée par un n-uplet de Studio, la contrainte sera violée mais le SGBD ne s'en apercevra pas :-(

Contraintes Globales

On distingue :

- Les contraintes restreignant les n-uplets d'une relation.

```
1) CREATE TABLE Star (
2)     nom CHAR(30) UNIQUE,
3)     adresse VARCHAR(255),
4)     sexe CHAR(1),
5)     datenaiss DATE,
6)     CHECK( sexe  = 'F' OR nom NOT LIKE 'Mr. %' )
);
```

  Même gestion que pour la vérification de valeurs d'attributs.

- Les assertions. Permettent d'exprimer des contraintes portant sur plusieurs relations

Assertions

Sont déclarées au niveau du schéma.

```
CREATE ASSERTION <nom> CHECK (<condition>)
```

La condition pour une assertion doit toujours être vraie.

Toute modification susceptible d'entrainer une violation sera rejetée par le SGBD.

Exemples

- Tout président de studio doit avoir un bénéfice net supérieur à 1000000.

```
CREATE ASSERTION RichePres CHECK
     (NOT EXISTS
            (SELECT *
              FROM Studio, Producteur
              WHERE Idpres = Idprod AND benefnet < 1000000)
     );
```

Triggers

- Permettent au programmeur (non plus au système) de déclencher des vérifications. BD Actives

- Font partie de la norme SQL3

- Bien des constructeurs les offrent au niveau SQL2.

Triggers : Principes de base

Règles <**`evenement`**> <**`condition`**> <**`action`**>

1. testés lorsque certains évenements (insert, update, delete) surviennent (spécifiés par le programmeur).

2. la condition est évaluée si faux rien ne se passe sinon

3. l'action associée est exécutée.

Triggers : Exemple

```
CREATE TRIGGER BenefNetTrigger
AFTER UPDATE OF benefnet ON Producteur
REFERENCING
      OLD AS Oldtuple,
      NEW AS Newtuple,
WHEN (Oldtuple.benefnet > Newtuple.benefnet)
      UPDATE Producteur
      SET benefnet = Oldtuple.benefnet
      WHERE IdProd = Newtuple.IdProd
    FOR EACH ROW
```

Triggers : Exemple

```
CREATE TRIGGER AvgBenefNetTrigger
INSTEAD OF UPDATE OF benefnet ON Producteur
REFERENCING
      OLDTABLE AS Oldtable,
      NEWTABLE AS Newtable,
WHEN (500000 <=
      (SELECT AVG(benefnet)
      FROM ((Producteur EXCEPT Oldtable) UNION Newtable)
        )
      DELETE FROM Producteur
     WHERE (nom, adresse, IdProd, benefnet) IN Oldtable;
     INSERT INTO Producteur
          (SELECT * FROM Newtable);
```