

VUES

Vues (Views)

- On se souvient de l'architecture ANSI-SPARC?
- Niveau Externe:
  - Masquer certaines données à certains utilisateurs
  - Offrir une modularité de l'accès à la base de données
  - Effectuer certaines requêtes plus facilement
  - Augmenter la sécurité

Vues

- Une vue est le résultat dynamique d'une ou plusieurs opérations sur des relations pour produire une nouvelle relation
- C'est également une relation virtuelle qui n'existe pas forcément dans la BD. On dit que la vue n'est pas physiquement matérialisée. A la place, la requête qui définit la vue est exécutée à chaque fois que la vue est référencée dans une requête
- Les modifications sur la table de base sont reflétées dans la vue

Instruction CREATE VIEW

```
CREATE VIEW ViewName [ (newColumnName [,...]) ]
 AS select
 [WITH [CASCADED | LOCAL] CHECK OPTION]
```

- On peut assigner un nouveau nom aux colonnes de la vue, sinon, les noms de la table initiale sont réutilisés
- Quand les colonnes sont spécifiées, on doit avoir le même nombre de colonnes que dans la sous-requête

Exemple 1

```
CREATE VIEW Manager3Staff
AS    SELECT *
      FROM Staff
      WHERE branchNo = 'B003';
```

Table 6.3 Data for view Manager3Staff.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

Exemple 2 avec jointure

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
 AS SELECT s.branchNo, s.staffNo, COUNT(*)
 FROM Staff s, PropertyForRent p
 WHERE s.staffNo = p.staffNo
 GROUP BY s.branchNo, s.staffNo;
```

Table 6.5 Data for view StaffPropCnt.

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

DROP VIEW

- `DROP VIEW ViewName [RESTRICT | CASCADE]`
  - Permet de supprimer la vue

- RESTRICT ne permet la suppression que si aucune autre vue ne dépend de la vue (comme on l'a vu, on peut créer une vue à partir d'une vue)
- CASCADE entraine une suppression en cascade
- Exemple: DROP VIEW Manager3Staff**;**

Vues

- Lorsqu'une requête accède à une vue, la vue est d'abord remplacée par sa définition, soit la requête qui la définit. On parle de déploiement de vues.
  - Aucun besoin de maintenir la correspondance avec les tables de la BD
  - Mais inefficace pour des vues définies par des requêtes complexes
- On accède à une vue avec un select standard. La vue remplace une table dans le select.

Exemple: Requête sur une vue

Compter le nombre de propriétés gérées par chaque membre de la branche B003.

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
AS SELECT s.branchNo, s.staffNo, COUNT(*)
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo;

SELECT staffNo, cnt
FROM StaffPropCnt
WHERE branchNo = 'B003'
ORDER BY staffNo;
```

Requête sur la vue

Mise à Jour de vues

- On peut aussi mettre à jour des vues
- il y a plusieurs restrictions sur la mise à jour des vues.
- Une mise à jour n'est pas possible si la requête qui définit la vue utilise:
  - Une jointure (une seule relation dans le FROM)
  - Une opération ensembliste (union, intersection, différence)
  - Une fonction d'agrégation
  - Le mot-clé DISTINCT
  - Un regroupement (GROUP BY)

Vues avec mise à jour

- On peut contrôler le comportement des vues qui peuvent être mises à jour au moyen de la commande [WITH [CASCADED | LOCAL] CHECK OPTION]
- Quand cette option est spécifiée, les opérations INSERT et UPDATE sur la vue doivent seulement accepter les nouveaux tuples qui respectent la définition de la vue. Sinon ces opérations seront rejetées

```
CREATE VIEW ViewName
    [ (newColumnName [,...]) ]
   AS select …
   [WITH [CASCADED | LOCAL] CHECK OPTION]
```


Exemple - WITH CHECK OPTION

```
CREATE VIEW Manager3Staff
AS      SELECT *
        FROM Staff
        WHERE branchNo = 'B003'
WITH CHECK OPTION;
```

- On ne peut pas mettre à jour le branchNo dans la vue car cela rendra les instances qui s'y trouvent non valides (elles sont toutes liées à B003).
- WITH CHECK OPTION permet de contrôler ce type de modification non conforme à la définition de la vue.

CASCADED et LOCAL

- CASCADED CHECK OPTION (équivalent à CHECK OPTION): Tient compte de la définition de toutes les vues dont la vue courante est dérivée
- LOCAL CHECK OPTION : ne respecte que les conditions de la vue « elle-même » sans tenir compte des vues de base dont elle est dérivée

Exemple

Soit les vues suivantes:

-
```
CREATE VIEW comedies AS
    SELECT *
    FROM films
    WHERE kind = 'Comedy';
```

Vue qui comprend tous les films de type comédie

-
```
CREATE VIEW universal_comedies AS
    SELECT *
    FROM comedies
    WHERE classification = 'U';
```

Vue qui comprend tous les films de type comédie et de classification U

Ref: http://www.postgresql.org/docs/9.4/static/sql-createview.html

Exercice

Créez la vue suivante dans MovieViews.sql:

```
CREATE VIEW comedies AS
    SELECT *
    FROM films
    WHERE kind = 'comedy';
```

- Insérez les données suivantes:
  - `INSERT INTO comedies (title, classification, kind) VALUES ('test1', 'U', 'comedy');`
  - `INSERT INTO comedies (title, classification, kind) VALUES ('test2', 'U', 'Drame');`
- Que constatez-vous?

Réponse

Exemple LOCAL CHECK OPTION

```
CREATE VIEW universal_comedies AS
    SELECT *
    FROM comedies
    WHERE classification = 'U'
    WITH LOCAL CHECK OPTION;
```

Local Check Option: Ici, si on insère un tuple dans la vue, on ne rejettera que les lignes qui ne correspondent pas à une classification U sans vérifier si le type est bien une comédie

Ref: http://www.postgresql.org/docs/9.4/static/sql-createview.html

Exercice

- Testez le code suivant:

```
CREATE VIEW universal_comedies AS
    SELECT *
    FROM comedies
    WHERE classification = 'U';
```

Notez que c'est une vue définie à partir d'une vue!

- --- AUCUN CHECK - TOUTES LES INSERTIONS FONCTIONNENT MEME SI ELLES NE LE DEVRAIENT PAS
  - `INSERT INTO universal_comedies (title, classification, kind) VALUES ('test1', 'U', 'comedy');`
  - `INSERT INTO universal_comedies (title, classification, kind) VALUES ('test1', 'U', 'drama');`
  - `INSERT INTO universal_comedies (title, classification, kind) VALUES ('test1', 'PG', 'comedy');`

Exercice suite

- Maintenant supprimez la vue :
- `DROP VIEW IF EXISTS universal_comedies  CASCADE;`
- Et incluez dans sa définition WITH LOCAL CHECK OPTION;
- Testez maintenant la vue et l'insertion des données.
- Que constatez-vous?

Réponse

Réponse suite

Exemple CASCADED CHECK OPTION

- ```
  CREATE VIEW pg_comedies AS
      SELECT *
      FROM comedies
      WHERE classification = 'PG'
      WITH CASCADED CHECK
      OPTION;
  ```

Cascaded Check Option: Ici si on insère un tuple dans la vue, on rejettera les lignes qui ne correspondent pas à une classification PG ET à une comédie!
  ```

Exercice

- Créez la vue pg_comedie

```
CREATE VIEW pg_comedies AS
    SELECT *
    FROM comedies
    WHERE classification = 'PG'
    WITH  CASCADED CHECK OPTION;
```

Que constatez-vous?

- et testez l'insertion suivante:
  - `INSERT INTO pg_comedies (title, classification, kind) VALUES ('test1', 'U', 'comedy');`
  - `INSERT INTO pg_comedies (title, classification, kind) VALUES ('test1', 'U', 'drama');`
  - `INSERT INTO pg_comedies (title, classification, kind) VALUES ('test1', 'PG', 'comedy');`

Réponse

FONCTIONS

Exercice

- Soit la base de données Hotel
- Exécutez les scripts HotelSchema et HotelData.
- On veut créer une fonction en PL/pgSQL qui retourne le nombre total d'enregistrements dans la table hotel

Réponse

TRIGGERS

Exercice: Carnet d'adresse (Row-level Trigger)

Soit le code SQL suivant:

```
DROP TABLE IF EXISTS addressbook CASCADE;
DROP TABLE IF EXISTS phonebook CASCADE;

CREATE TABLE addressbook (
id SERIAL Primary Key,
name VARCHAR(10),
address VARCHAR(20),
phoneNum VARCHAR(10));

CREATE TABLE phonebook (
id SERIAL Primary Key,
name VARCHAR(10),
phoneNum VARCHAR(10));
```

Quand on insère des données dans addressbook, on veut insérer automatiquement certaines de ces données dans phonebook au moyen d'un trigger

1. Créez un trigger qui appelle une fonction add_to_phonebook() lors d'une insertion de données sur la table addressbook
2. Créez la fonction add_to_phonebook() qui ajoute le nom et le numéro de tel inséré dans phonebook(utilisez la variable new)
3. Testez le trigger avec une insertion dans addressBook

Réponse: Création du trigger

Réponse: Fonction

Réponse: Insertion de données

Exercice Hotel

- Dans votre base de donnéees Hotel, créez un trigger lors de l'insertion d'une nouvelle chambre dans Room pour les situations suivantes:
- Trigger 1 : Le prix des chambres doubles doit être supérieur à 100 $. Sinon vous devez lancer une exception
- Trigger 2: Le prix des chambres doubles doit être supérieur au prix de la chambre simple la plus coûteuse. Sinon vous devez lancer une exception

Réponse : Trigger 1 - Fonction

Réponse: Trigger 1

Réponse: Trigger 2 - Fonction

Réponse: Trigger 2