SGBD : Programmation et administration des bases de données [M2106]

Hocine ABIR

10 mars 2014

IUT Villetaneuse
E-mail: abir@iutv.univ-paris13.fr

TABLE DES MATIÈRES

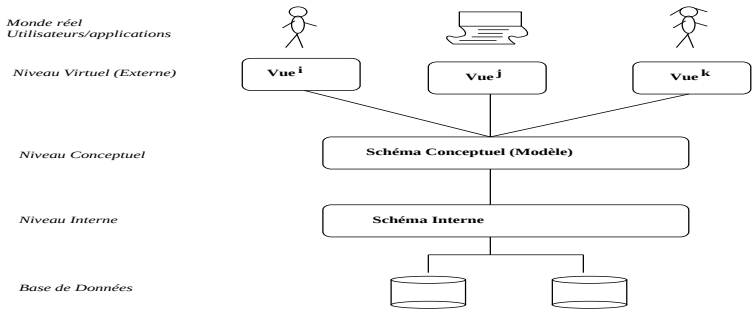
CHAPITRE

CINQ

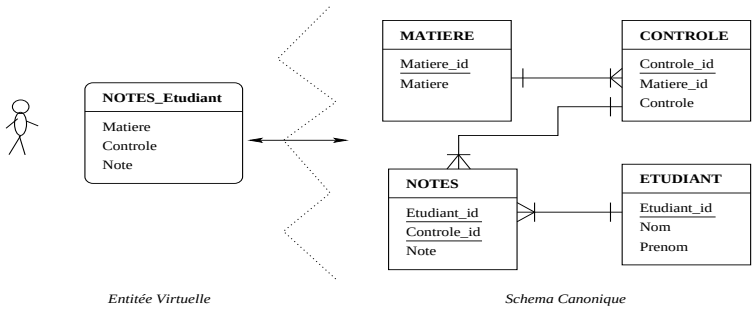
Vues et modèle externe de données

5.1 Introduction

Le modèle externe est constitué d'une (ou plusieurs) vues(s) du modèle de base. Il a pour objectif de populariser le modèle de base, c'est à dire de l'adapter :
- aux différents utilisateurs (ou groupe d'utilisateurs),
- aux différentes applications,
- aux besoins de gestion et d'administration des données.

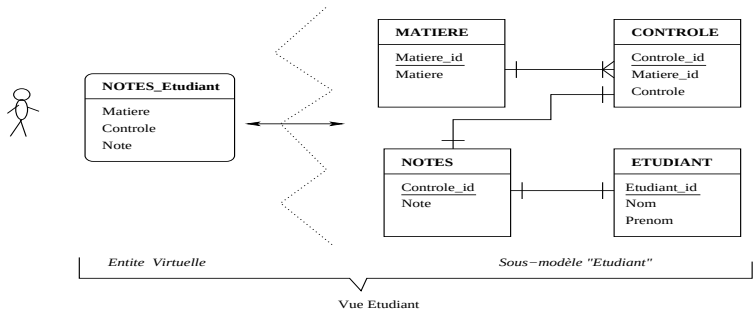


Ces caractéristiques impliquent que le modèle externe doit être capable :
- de restituer une entité (relation) virtuelle *dé-normalisée* des données : *vue* ,
- de restreindre horizontalement et verticalement les instances de ces entités : *sécurité*, *confidentialité*
- de générer de nouvelles données : *données dérivées*.



5.2 Vues

5.2.1 Exemple



Le sous-modèle *"Etudiant"* apparait comme un modèle de données propres à chaque étudiant.

5.2.2 Définition

Une vue est une relation (entité) virtuelle engendrée à partir du modèle canonique par re-composition des relations de base (jointure, projection, etc).

Exemple :

$$Notes\_Etudiant(Matiere, Controle, Note) =$$
$$(\Pi^{[Matiere,Controle,Note]}$$
$$(Matiere \bowtie Controle \bowtie Notes \bowtie (\sigma^{[Etudiant\_id=utilisateur]} Etudiant)))$$

5.2.3 Mise oeuvre

Schéma

```
CREATE TABLE etudiant
(
   etudiant_id
varchar primary key,
   nom     varchar,
   prenom  varchar
);

CREATE TABLE Matiere
(
   matiere_id   varchar primary key,
   matiere      varchar

);
```

```
CREATE TABLE Controle
(
    controle_id    int primary key,
   matiere_id
varchar references matiere,
   Controle    varchar
);

CREATE TABLE Notes
(
  etudiant_id varchar
        references  etudiant  ,
  controle_id int  references  controle ,
  note decimal(4,2),
   primary key(etudiant_id,controle_id)
);
```

Instance

```
=> select * from etudiant;
 etudiant_id |   nom    | prenom
-------------+----------+--------
 toto        | GAILLARD | Pierre
 titi        | LECOQ    | Paul
(2 rows)

=> select * from matiere;
 matiere_id |       matiere
------------+--------------------
 M2106      | Programmation et
            |    administration des BD
 M3106C     | Bases de données avancées
(2 rows)
```

```
=> select * from Controle;
 controle_id | matiere_id |    controle
-------------+------------+----------------
          10 | M2106      | Controle Moyen
          20 | M3106C     | Controle Court
          30 | M2106      | Controle Long
(3 rows)

=> select * from notes;
 etudiant_id | controle_id | note
-------------+-------------+-------
 toto        |          10 | 12.50
 titi        |          10 | 11.00
 toto        |          30 | 15.50
 titi        |          20 | 13.00
(4 rows)
```

Vue

```
1 CREATE view notes_etudiant AS
2 SELECT  Matiere,Controle,Note
3    FROM Matiere natural join Controle
4     natural join Notes natural join Etudiant
5    WHERE etudiant_id=current_user;
6
7 grant select on notes_etudiant to  titi,toto;
```

```
 (toto) [demodb] => select * from notes;
ERROR:  permission denied for relation notes
(toto) [demodb] => select * from notes_etudiant;
                matiere                |    controle    | note
---------------------------------------+----------------+-------
 Programmation et administration des BD | Controle Moyen | 12.50
 Programmation et administration des BD | Controle Long  | 15.50
 (2 rows)
```

```
 (titi) [demodb] => select * from notes;
ERROR:  permission denied for relation notes
(titi) [demodb] => select * from notes_etudiant;
                matiere                |    controle    | note
---------------------------------------+----------------+-------
 Programmation et administration des BD | Controle Moyen | 11.00
 Bases de données avancées              | Controle Court | 13.00
(2 rows)
```

5.3 Commande CREATE VIEW

```
CREATE [ OR REPLACE ] VIEW nom_vue
        [ (
              column_name [, ...]
        ) ]
    AS requete_select
```

- crée une table fictive (sans espace mémoire) et
- associe une règle ON SELECT à cette table fictive.

Le système n'autorise pas de mises à jour de vue puisqu'il s'agit d'une table fictive.

```
=>   SELECT  pg_get_ruledef(pg_rewrite.oid) as "Définition"
 FROM pg_rewrite,pg_class,pg_user
 WHERE ev_class=pg_class.oid
 and pg_user.usesysid=pg_class.relowner
 and relname='notes_etudiant'
 and pg_user.usename='abir';

 Définition
----------------------------------------------------------------------------------
 CREATE RULE "_RETURN" AS ON SELECT TO notes_etudiant
  DO INSTEAD
    SELECT matiere.matiere, controle.controle, notes.note
    FROM (((matiere NATURAL JOIN controle) NATURAL JOIN notes)
    NATURAL JOIN etudiant)
   WHERE ((notes.etudiant_id)::name = "current_user"());
(1 row)
```

```
CREATE TABLE r_notes_etudiant
(
  matiere varchar,
  controle varchar,
  note decimal(4,2)
);

 CREATE RULE "_RETURN" AS ON SELECT TO r_notes_etudiant
  DO INSTEAD
    SELECT matiere, controle, note
    FROM (((matiere NATURAL JOIN controle) NATURAL JOIN notes)
    NATURAL JOIN etudiant)
   WHERE ((notes.etudiant_id)::name = "current_user"());
```

5.4 Règles

5.4.1 Structure

```
CREATE [ OR REPLACE ] RULE nom
    AS ON evenement
    TO table [ WHERE condition ]
    DO [ INSTEAD ]
      { NOTHING |
        commande |
```

```
7       ( commande ; commande ... )
8     }
```

- `nom` : désigne le nom de la règle à créer.
- `evenement` : peut être `SELECT`, `UPDATE`, `DELETE` ou `INSERT`.
- `objet` : désigne une table ou une vue (ou une colonne d'une table [non implémentée]).
- `condition` : représente une expression SQL logique. L'expression d'une condition ne peut faire référence à une table sauf `new` et `old`.
- `commande` peut être une des alternatives suivantes :

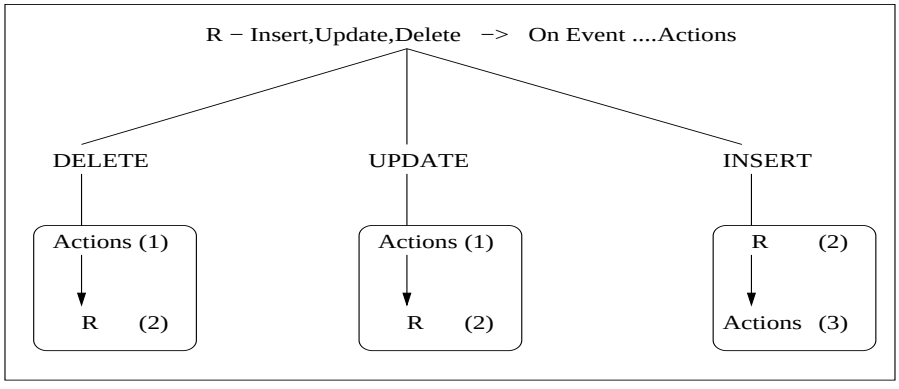
```
NOTHING |
requête   |
(  requête; requête ... )
```

  où `requête` est une commande SQL `SELECT`, `INSERT`, `UPDATE`, `DELETE`, ou `NOTIFY`.
- la `condition` et l'`action` peuvent utiliser les noms spéciaux de table `new` et `old` pour référencier les données dans la table referenciée par l'`objet`.
  - `new` : est valide pour des règles `ON INSERT` et `ON UPDATE` (`événement`) pour référencier le nouveau tuple inséré ou mis à jour.
  - `old` : est valide pour des règle `ON UPDATE` et `ON DELETE` (`événement`) pour référencier un tuple existant mis à jour ou supprimé.

5.4.2 Description

- le système de règle permet :
  - de définir des actions alternatives à effectuer pour des commandes d'insertions, de mise à jour, ou de suppression sur des tables.
  - d'implémenter des vues sur des tables.

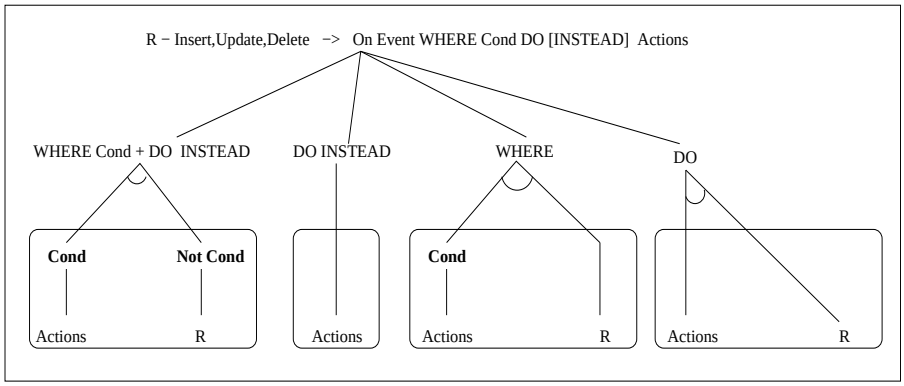
- Chaque tuple lu, inséré, mis à jour, ou supprimé, a :
  - une instance `old` pour des requêtes `SELECT`s, `UPDATE`s et `DELETE`s
  - une instance `new` pour des requêtes `INSERT`s and `UPDATE`s.
- Toutes les règles pour un type événement donné et un objet donné (table, vues, ...) sont examinées dans un ordre indéfini.
- Si la condition specifiée dans la clause `WHERE` éventuelle est vraie, la partie action de la règle est exécutée.
- L'action est effectuée :
  - à la place de la requête d'origine si `INSTEAD` est spécifié,
  - après la requête d'origine si règle `ON INSERT`,
  - avant la requête d'origine si règle `ON UPDATE` ou `ON DELETE`.



- Dans la `condition` et l'`action` de la règle, les valeurs des champs de l'instance `old` et/ou l'instance `new` sont substituées pour les références `old.nom-attribut` et `new.nom-attribut`.

- La partie `action` peut comporter :
  - aucune requête si `NOTHING` est spécifié comme action,
  - une requête,
  - plusieurs requêtes regroupées par des parenthèses. Ces requêtes sont exécutées dans l'ordre dans lequel elles sont spécifiées dans l'`action`.

- L'`action` de la règle est exécutée avec le même identifiant de commande et de transaction que la commande qui a déclenché la règle.



5.5 Règles dans le Catalogue

- `pg_rewrite`
  Contient la description des règles de reécriture de tables et vues.

`rulename` : nom de la règle.

`ev_type` : type d'événement associé à la règle : '1' = SELECT, '2' = UPDATE, '3' = INSERT, '4' = DELETE.

`ev_class` : la table sur laquelle la règle porte.

`ev_attr` : le numero (rang) de l'attribut sur lequel porte la règle (0 : toute la table).*

`is_instead` : vrai si règle `INSTEAD`.

`ev_qual` : condition de la règle.

`ev_action` : action associé à la règle.

- `pg_class`
  Contient une entrée pour chaque objet : table,index,séquence,vues, etc.

`oid` : oid de la relation

`relname` : nom de la relation (voir `relkind`).

`relowner` : référence (usesysid) au propriétaire de la relation dans `pg_shadow`

`relhasrules` : "true" si la table possède aumoins une règle

5.6 Vues matérialisées

En pratique, les données sont souvent dé-normalisées à divers degrès. La raison essentielle de cette pratique est l'efficacité et la simplicité : les données dé-normalisées nécessitent moins de jointure donc plus rapide à consulter.

En contre partie, la mise à jour des données dé-normalisées est plus complexe. Des contraintes supplémentaires (DFs) sont nécessaires et leur vérification peut compromettre ce gain de performance.

Les Vues Matérialisées sont un exemple de ce type de dé-normalisation. Une Vue Matérialisée est une entité dont les instances sont dérivées des entités du modèle de base par une *requête* d'interrogation : la notion de Vue Matérialisée est donc fortement lié à la notion de réplique (càd de dé-normalisation) : Comment et Quand mettre à jour les données répliquées ? Autrement dit :

- Quelle est la date de préemption des copies : Jusqu'à quand les données copiées sont encore valides ?
- Combien ça coute de les mettre à jour :

C'est sur ces deux questions que l'on va distinguer plusieurs façons de gérer les Vues Matérialisées.

5.6.1 Snapshot

Les mises à jour de la Vue Matérialsée sont effectuées périodiquement. On dit que la Vue Matérialisée est raffraichie (`refresh`) : la méthode peut être brutale.

5.6.2 Eager

Les mises à jour de la Vue Matérialsée sont effectuées en même temps que celles des entités dont elle est dérivée.

5.6.3 Lazy

Les mises à jour de la Vue Matérialsée sont effectuées à la fin de chaque transaction (`COMMIT`).

5.6.4 Very Lazy

la Vue Matérialisée `Very Lazy` est analogue à `Snapshot`, sauf les mises à jour sont enregistrés de manière incrémentale (analogue à `Historique`) puis appliquées périodiquent (au moment du raffraichissement).