Bases de Données

Vues

Arnaud Delhay-Lorrain

Département Informatique
IUT de Lannion
Université de Rennes

BUT Informatique

Plan du cours

Introduction

- jusqu'à maintenant : tables de bases permanentes
  - elles existent jusqu'à destruction explicite par un `DROP TABLE`.
- existence d'une notion de schéma externe, correspondant au concept de tabels dérivées :
  - les photographies (snapshot)
  - les vues

Introduction (2)

Definition (Les photographies)

sont des tables contenant des données déduites à partir des tables de base. Elles sont stockées physiquement dans la base.

- perte d'espace de stockage.
- difficultés de maintenir la cohérence avec les tables de base dont elles sont issues.

En pratique, avec SQL2, on crée des tables temporaires

- `CREATE TABLE ... TEMPORARY`
- vides en début de session SQL
- effacées automatiquement à la f in de chaque session
- peuvent être effacées à chaque `COMMIT`, et modifiées même au sein d'une transaction en lecture seule

Introduction (3)

Definition (Une vue)

est une table dérivée dynamique, véritable fenêtre sur la base de données

- pas stockée physiquement dans la base : pas de problème de cohérence
- évaluée au moment de la consultation : ralentissement des traitements

Plan : 2 - Définition des vues

Définition des vues

Tuorial D

```
VAR PERSONNEL VIRTUAL
        <requete>
```

SQL

```
CREATE VIEW <nom_vue> [(liste_attributs)]
AS <expression_de_selection>;
```

- Définition d'une table virtuelle à partir des tables présentes dans la clause `FROM`. Ces tables sont appelées table sources. Elles peuvent être des tables de base ou de vues.
- Aucune extraction de n-uplet au moment du `CREATE VIEW`.

Plan : 2 - Définition des vues

Exemple de vue (1)

Vue reprenant uniquement les élèves de première année

```
CREATE VIEW eleve_de_premiere_annee
        (num_eleve, prenom, date_de_naissance)
AS SELECT *
   FROM eleves
   WHERE annee = 1;
```

Ou en gardant les noms des attributs d'origine

```
CREATE VIEW eleve_de_premiere_annee
AS SELECT *
   FROM eleves
   WHERE annee = 1;
```

Exemple de vue (2)

Le nom des élèves de première année pratiquant le surf de niveau 2

```
SELECT eleves_de_premiere_annee.nom
FROM eleves_de_premiere_annee
INNER JOIN activites_pratiquees
ON eleves_de_premiere_annee.num_eleve =
              activites_pratiquees.num_eleve
WHERE niveau = 2
  AND activites_pratiquees.nom = 'surf';
```

Plan : 2 - Définition des vues

Mécanisme d'évaluation des vues

Mécanisme d'évaluation :

- Recherche dans la méta-base de la définition de la vue
- Transformation de la requête initiale pour injecter la définition de la vue dans la requête. On appelle cela la composition de vues
- Exécution de la requête composée

Remarques :

- Utilisation immmédate d'une vue dès sa création
- Rien ne permet à un utilisateur des distinguer une vue d'une table de base
- Toute modification dans la base se répercute dans les vues (fenêtre dynamique)

Raison d'être des vues

- Augmenter l'indépendance logique
- Préserver la confidentialité
- Simplifier les requêtes ou masquer la complexité

Plan : 3 - Raisons d'être des vues

Augmenter l'indépendance logique

De quoi s'agit-il ? Indépendance de l'utilisateur vis à vis de l'organisation logique globale des données
Exemple :

```
PROJET(nom_projet, code_activite,
       tarif, heures_prevues)
```

+ vue contenant le prix de revient :

```
CREATE OR REPLACE VIEW projet_prix
      (nom_projet, code_activite, tarif,
       heures_prevues, prix)
AS SELECT projet.*, tarif * heures_prevues
   FROM projet;
```

Augmenter l'indépendance logique

Modification du schéma (redondance évitée)

```
code_activite --> tarif

DESCRIPTION_PROJET(nom_projet,code_activite,
                   heures_prevues)

TABLE_DES_TARIFS(code_activite,tarif)
```

Augmenter l'indépendance logique

Nouvelle vue :

```
CREATE OR REPLACE VIEW projet_prix
       (nom_projet, code_activite, tarif,
        heures_prevues, prix)
AS SELECT nom_projet, description_projet,
          code_activite, tarif,
          tarif * heures_prevues
   FROM description_projet d INNER JOIN
        table_des_tarifs t
   ON d.num_code_activite = t.num_code_activite;
```

Plan : 3 - Raisons d'être des vues

Préserver la confidentialité

Exemple :

```
EMPLOYE(numero, nom, prenom, adresse,
        num_service, salaire, performance);
```

Hypothèses :

- aucun utilisateur ne possède de privilège sur la table `EMPLOYE`
- la table est dans le schéma de l'utilisateur `del`

Contrôle dépendant du contexte

Chaque utilisateur peut visualiser un seul n-uplet de la relation : celui qui le concerne.

```
CREATE OR REPLACE VIEW infos_me_concernant
AS SELECT *
   FROM employe
   WHERE UPPER(nom)=UPPER(USER);

GRANT SELECT ON infos_me_concernant
TO grouprole;
```

Contrôle dépendant du contexte et du contenant

Chaque utilisateur peut visualiser un seul n-uplet de la relation : celui qui le concerne. Il peut aussi modifier son adresse.

```
CREATE OR REPLACE VIEW infos_me_concernant
AS SELECT *
   FROM employe
   WHERE UPPER(nom)=UPPER(USER);

GRANT SELECT, UPDATE OF adresse
ON infos_me_concernant
TO grouprole;
```

Plan : 3 - Raisons d'être des vues

3 Raisons d'être des vues
- Augmenter l'indépendance logique
- Préserver la confidentialité
- Simplifier les requêtes

Simplifier les requêtes

- Quand elles concernent de multiples jointures et sont fortement imbriquées
- Quand elles portent sur des tables aux multiples colonnes

Plan : 4 - Mise à jour au travers des vues

Introduction au problème

Il n'est pas toujours possible, voire sensé de faire des mises à jour au travers de vues.

- Il faut pouvoir propager les modifications vers les tables sources
- Idéal : relation bi-univoque entre n-uplets de la vue et ceux de la table source

Exemples de vues non-modifiables :

- ```
  CREATE VIEW total_des_points(num_eleve,total)
  AS SELECT num_eleve, SUM(points)
  FROM resultats
  GROUP BY num_eleve;
  ```
- Vues construites à partir de plusieurs tables sources
  ```

Plan : 4 - Mise à jour au travers des vues

Conditions pour SQL

Conditions de SQL2, SQL89 pour qu'une une vue soit "modifiable" :

- La définition de la vue ne doit pas contenir de `JOIN`, `UNION`, `INTERSECT` ou `EXCEPT`.
- La clause `SELECT` de la définition de la vue ne doit pas contenir de `DISTINCT`
- La claude `SELECT` ne peut contenir que des références aux attributs de la table source (pas de `SUM`, `AVG`, `COUNT`, ...)
- La clause `FROM` contient exactement une table ou une vue elle-même modifiable
- La clause `WHERE` ne peut pas contenir une sous-question dont la clause `FROM` prossède une référence à la même table que celle de la caluse `FROM` de la requête principale (exemple : définition des poids légers à partir de la moyenne des poids)
- La définition de la vue ne contient pas de clause `GROUP BY` ou `HAVING`

Conditions pour SQL

Une vue ne satisfaisant pas les conditions citées ci-avant est dite en “lecture seule” (READ ONLY).

Remarque : La clause `WITH CHECK OPTION` empèche que l’utilisateur ajoute ou modifie dans une vue des n-uplets non-conformes à la définition de la vue.
Exemple : Gros salaires (> 100 000) et on ajoute un salaire à 50 000 au travers de la vue.

Plan : 4 - Mise à jour au travers des vues

Conditions pour SQL

- Possibilité avec ORACLE sous certaines conditions
- IMPOSSIBLE sous PostgreSQL : on utilisera les triggers (voir cours sur les triggers)