Systèmes de Gestion de Bases de Données - 2e

Chapitre 7 - Les vues

Daniel Schreurs

27 octobre 2021

Haute École de Province de Liège

Table des matières du chapitre i

Introduction

Table des matières de la section : Introduction i

Introduction : Définition

- Jusqu'à présent, nous avons parlé des **tables de base permanentes**.
- Une **table de base** existe réellement : il lui correspond des enregistrements stockés sur le support physique contenant la base.
- **Permanent** indique que, dès que la table est créée, elle persiste dans la base de données jusqu'à ce qu'elle soit explicitement effacée au moyen de la commande `DROP TABLE`

Introduction : Définition

Dans les SGBD relationnels, la notion de schéma externe correspond au concept de table dérivée. On distingue 2 types de tables dérivées :

- les **photographies**
- les **vues**.

Introduction : Définition

Une photographie est une table contenant des données déduites à partir des tables de base (ou même d'autres photographies).
Une photographie est stockée physiquement dans la base. Deux inconvénients :

- Perte d'espace de stockage
- Difficulté de maintenir la cohérence entre la photographie et les tables de base du fait de la redondance

Introduction : Définition

La norme SQL2 définit le concept de tables temporaires.
Cela convient, par exemple, aux applications qui ont besoin de tables pour contenir des résultats intermédiaires pendant un petit intervalle de temps.
Les tables temporaires sont privées à l'application qui les utilise : donc, pas d'accès concurrents. Caractéristiques essentielles :

- Elles se créent en spécifiant TEMPORARY dans la commande `CREATE TABLE`
- Elles sont toujours vides au début d'un session SQL
- Elles peuvent éventuellement être vidées à chaque `COMMIT`
- Elles sont effacées automatiquement à la fin de chaque session
- Elles peuvent être modifiées même au sein d'une transaction en lecture seule

Introduction : Définition

- Une vue est une table dérivée dynamique.
- Elle n'est pas stockée physiquement dans la base.
- Le problème de cohérence ne se pose donc pas.
- Par contre, elle peut entraîner un ralentissement des traitements si la vue doit être générée souvent.

Définition des vues

Définition des vues : Syntaxe

Syntaxe pour la création d'une vue

```
1 Créer_vue ::=
2   CREATE VIEW nom_vue [ ( Liste_colonne ) ]
3   AS expression_de_sélection
4   [ WITH CHECK OPTION];
```

Définition des vues : Syntaxe

- Cette commande définit **une table virtuelle** à partir des tables de la clause `FROM` de l'`expression_de_sélection`. Les tables présentes dans la clause `FROM` sont appelées tables sources. Il peut s'agir de tables de base ou de vues. La commande CREATE VIEW n'extrait aucune ligne de la base, elle inscrit la définition de la vue dans les tables de la méta-base (information schema) : `TABLES`, `VIEWS`, `VIEW_TABLE_USAGE` et `VIEW_COLUMN_USAGE`.
- La commande `CREATE VIEW` **n'extrait aucune ligne de la base**, elle inscrit la définition de la vue dans les tables de la méta-base (information schema) : `TABLES`, `VIEWS`, `VIEW_TABLE_USAGE` et `VIEW_COLUMN_USAGE`.

Définition des vues : Exemples

Exemple : Vue reprenant uniquement les étudiants de 1ère année

```
1 CREATE or replace VIEW eleves_de_premiere
2             (num_eleve, nom, prenom, date_naissance,
3              poids, annee)
4 AS
5 SELECT *
6 FROM eleves
7 WHERE annee = 1;
```

Définition des vues : Exemples

- Dès qu'une vue est définie, on peut l'utiliser comme s'il s'agissait d'une table de base.
- **Rien ne permet à un utilisateur de faire la distinction entre une table de base et une vue.**
- Cependant, il faut émettre des restrictions pour les requêtes de mises à jour sur les vues.

Définition des vues : Exemples

Exemple :Rechercher le nom des élèves de première année pratiquant le surf au niveau 1

```sql
SELECT ep.nom
FROM eleves_de_premiere ep,
     activites_pratiquees ap
WHERE ep.num_eleve = ap.num_eleve
  AND niveau = 1
  AND ap.nom = 'surf';
```

Définition des vues : Exemples

Afin de répondre à cette requête, le SGBD va d'abord rechercher dans la méta-base la définition de la vue `eleves_de_premiere`. Il transforme alors la requête initiale pour obtenir

Exemple :Rechercher le nom des élèves de première année pratiquant le surf au niveau 1

```sql
SELECT eleves.nom
FROM eleves,
     activites_pratiquees ap
WHERE eleves.num_eleve = ap.num_eleve
  AND niveau = 1
  AND ap.nom = 'surf'
  -- Ici le prédicat en plus
  AND annee = 1;
```

Définition des vues : Exemples

Cette transformation est appelée composition de vues. L'appellation **fenêtre dynamique sur la base** se justifie par le fait que toutes **les modifications effectuées sur la ou les tables sources sont automatiquement reportées au travers de la vue**.

Raisons d’être des vues

Raisons d’être des vues

Une vue peut être utile pour :

- Masquer la complexité du schéma de la base
- Aider à la formulation d’une requête
- Jouer le rôle de table intermédiaire dans la résolution d’une requête complexe nécessitant plusieurs étapes

Rôles plus profonds :

- **Augmenter l’indépendance logique** en définissant des schémas externes
- **Préserver la confidentialité des données**

Raisons d'être des vues : Augmenter l'indépendance logique

Exemple (suite) : Et une vue contenant - en plus - pour chaque projet le prix de revient

```
1 CREATE VIEW projets_prix
2             (nom_projet, code_activite, tarif,
3              heures_prevues, prix)
4 AS
5 SELECT projets.*,
6        tarif * heures_prevues
7 FROM projets;
```

Raisons d'être des vues : Augmenter l'indépendance logique

Exemple (suite) : Pour que les programmes ne souffrent pas de cette modification il suffit de modifier la définition de la vue en conservant les mêmes noms pour la vue et ses colonnes

```
1 CREATE VIEW projets_prix
2   (nom_projet, code_activite, tarif,
3    heures_prevues, prix)
4 AS SELECT nom_Projet, dp.code_activite, tarif,
5           heures_prevues, tarif * heures_prevues
6    FROM description_projets dp, table_tarifs tf
7    WHERE dp.code_activite = tf.code_activite;
```

Raisons d’être des vues : Augmenter l’indépendance logique

Exemple (suite) : Pour éviter de la redondance si le tarif est lié au code activité on peut changer le schéma logique de la base vers le schéma équivalent

Raisons d’être des vues : Augmenter l’indépendance logique

- En conservant les mêmes noms pour la vue et ses colonnes, on ne doit pas changer les requêtes et donc les programmes d’applications qui l’utilisent.
- Les programmes sont dès lors insensibles aux modifications logiques de la base de données

Raisons d’être des vues : Préserver la confidentialité

- Les vues permettent d’assurer la confidentialité des données.
- Il est possible d’interdire l’accès à certaines lignes (contenu) et/ou à certaines colonnes (contenant) d’une table.
- Grâce aux vues, il est possible d’émettre des restrictions d’accès en fonction du contexte.

Raisons d'être des vues : Préserver la confidentialité

Quelques exemples vont permettre d'illustrer cette notion. Ces exemples seront basés sur la table `EMPLOYES` suivante :

```
EMPLOYES (Numero, nom, prenom, adresse, num_service
, salaire, performance
```

Nous supposons qu'aucun des utilisateurs cités dans les exemples ne possède de privilèges sur la table `EMPLOYES` et que cette dernière a été créée dans le schéma del.

Raisons d'être des vues : Préserver la confidentialité

Quelques exemples vont permettre d'illustrer cette notion. Ces exemples seront basés sur la table `EMPLOYES` suivante :

```
EMPLOYES (Numero, nom, prenom, adresse, num_service
, salaire, performance
```

Nous supposons qu'aucun des utilisateurs cités dans les exemples ne possède de privilèges sur la table `EMPLOYES` et que cette dernière a été créée dans le schéma del.

Raisons d'être des vues : Préserver la confidentialité

Contrôle dépendant du contexte

Chaque utilisateur peut visualiser un seul tuple de la relation : celui qui le concerne.

```
1  CREATE VIEW information_me_concernant
2  AS SELECT *
3     FROM employes
4     WHERE upper(nom) = upper(USER);
5
6  GRANT SELECT ON information_me_concernant
7  TO PUBLIC;
8
9  CREATE PUBLIC SYNONYM information_me_concernant
10 FOR del.information_me_concernant;
```

Raisons d'être des vues : Préserver la confidentialité

Contrôle dépendant du contexte et du contenant

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

```
1  CREATE VIEW information_me_concernant
2  AS SELECT *
3     FROM employes
4     WHERE upper(nom) = upper(USER);
5
6  GRANT SELECT, UPDATE (adresse) ON information_me_concernant
7  TO PUBLIC;
8
9  CREATE PUBLIC SYNONYM information_me_concernant
10 FOR del.information_me_concernant;
```

Raisons d'être des vues : Préserver la confidentialité

Contrôle dépendant du contexte et du contenant

L'utilisateur Astérix ne peut connaître ni le nom ni le prénom ni l'adresse des employés gagnant plus de 3000 mais il peut connaître toutes les autres informations

```
1 CREATE VIEW employes_riches
2 AS SELECT numero, salaire, performance
3    FROM employes
4    WHERE salaire > 3000;
5
6 GRANT SELECT ON employes_riches TO Asterix;
```

Raisons d'être des vues : Préserver la confidentialité

Contrôle dépendant du contexte et du contenant

L'utilisateur Panoramix peut visualiser le contenu de la table employes sauf le numero le nom le prénom et l'adresse des employés gagnant plus de 3000€

```
1 CREATE VIEW employes_riches
2   (numero, nom, prenom, adresse, num_service,
3    salaire, performance)
4 AS SELECT 'xxx', 'xxx', 'xxx', 'xxx',
5           num_service, salaire, performance
6    FROM employes
7    WHERE salaire > 3000;
```

Raisons d'être des vues : Préserver la confidentialité

Les utilisateurs Astérix, Panoramix et Abraracourcix sont responsables d'un service. Ils peuvent consulter toutes les informations des employés de leur service. On suppose qu'il existe une relation `SERVICES (num_service, #num_chef)` La vue de sécurité va être construite avec une jointure.

Raisons d'être des vues : Préserver la confidentialité

Contrôle dépendant du contexte et du contenant

L'utilisateur Panoramix peut visualiser le contenu de la table employes sauf le numero le nom le prénom et l'adresse des employés gagnant plus de 3000€

```
1  CREATE VIEW info_par_service
2  AS SELECT employes.*
3     FROM employes e, services s
4     WHERE e.num_service = s.num_service
5       AND s.num_chef =
6         (SELECT numero
7          FROM employes
8          WHERE upper(nom) = upper(USER));
9
10 GRANT SELECT ON info_par_service
11 TO Asterix, Panoramix, Abraracourcix;
```

Raisons d'être des vues : Préserver la confidentialité

Contrôle dépendant du contexte et du contenant

on souhaite que chaque élève puisse consulter sa moyenne sans pour autant connaître le détail des différentes cotes.

```
1  CREATE VIEW moyenne_par_eleve (nom, moyenne)
2  AS SELECT e.nom, AVG (points)
3     FROM eleves e, resultats r
4     WHERE e.num_eleve = r.num_eleve
5     GROUP BY e.num_eleve, e.nom
6     HAVING upper(e.nom) = upper(USER);
7
8  GRANT SELECT ON moyenne_par_eleve TO PUBLIC;
9
10 CREATE PUBLIC SYNONYM moyenne_par_eleve
11 FOR del.moyenne_par_eleve;
```

Mises à jour au travers des vues

Mises à jour au travers des vues

Il n'est pas toujours possible d'effectuer des mises à jour au travers des vues. [1]

1. Si la vue est telle qu'il existe une correspondance biunivoque entre les lignes de la vue et celle d'une seule table source, les modifications et les suppressions ne posent pas de problèmes et peuvent être propagées.

Mises à jour au travers des vues

SQL2 a défini un ensemble de conditions que doit respecter une vue pour être modifiable :

- La définition ne peut contenir les mots réservés : `JOIN`, `UNION`, `INTERSECT` ou `EXCEPT` (une seule table source)
- La clause `SELECT` ne peut contenir `DISTINCT`
- La clause `SELECT` ne peut contenir que des références aux colonnes de la table source (pas de `SUM`, `COUNT`,..)
- La clause `FROM` contient une seule table ou une vue elle-même modifiable
- La clause `WHERE` ne peut contenir une sous-question dont la clause `FROM` possède une référence à la table de la requête principale
- La vue ne peut contenir de `GROUP BY` ou `HAVING`

Les possibilités d’Oracle

Les possibilités d'Oracle

Oracle possède quelques possibilités supplémentaires dans le domaine des vues. Depuis Oracle 7.3, une vue dont la clause FROM de la définition contient plusieurs tables, est modifiable à condition de ne pas comporter :

- `DISTINCT`
- Des fonctions de calculs : `AVG`, `COUNT`, `MAX`, `MIN`, `SUM`
- Des opérateurs ensemblistes : `UNION`, `UNION ALL`, `INTERSECT`, `MINUS`
- `GROUP BY`, `HAVING`
- `START WITH`, `CONNECT BY`
- `ROWNUM`