BASE DE DONNEES ET SYSTEMES DE GESTION DE BASE DE DONNEES

(SGBD)

CHAPITRE IX : GESTION DES VUES ET DES INDEX

SOMMAIRE

DEFINITION ET MANIPULATION DES VUES

Les vues représentent le niveau d'abstraction immédiatement supérieur au niveau du schéma conceptuel de la base de données. Elles offrent à l'utilisateur un moyen d'améliorer l'indépendance logique des données.

On peut donc définir une vue comme un sous-ensemble d'une base de données, ce sous-ensemble étant constitué d'éléments présents dans la base et d'éléments dérivés, c'est-à-dire calculés au départ des éléments de la base de données.

Plus précisément, pour le modèle relationnel, une vue est perçue comme une table virtuelle issue de la projection d'un ensemble de tables. On appellera "source de la vue" un tel ensemble de tables.

Il importe ici de préciser ce que l'on entend par table "viruelle". La vue n'est pas enregistrée physiquement dans la base de données : seule sa définition est conservée. Lors de l'exécution d'une requête portant sur une vue, le SGBD effectue une opération de reconstitution de la définition de la vue.

CREATION D'UNE VUE

L'instruction permettant de créer une vue suit la syntaxe :

```
CREATE VIEW nom_vue [ (nom_col 1, nom_col 2, …) ]  AS SELECT   [ WITH CHECK OPTION ]
```

Le "select" est en fait une opération SELECT… FROM… WHERE… telle qu'elle a été décrite dans les chapitres précédents. Cependant, aucun tuple n'est extrait de la base de données par la commande CREATE VIEW.

La clause WITH CHECK OPTION conduit à une vérification des opérations INSERT et UPDATE réalisées sur la vue ; si la définition de la vue n'est pas respectée, l'opération sera refusée. Si la clause est omise, aucune vérification n'aura lieu.

Les noms des colonnes de la vue devront être précisés si un risque d'ambiguïté subsiste. Si, par exemple la vue résulte de la jointure de deux tables ayant un nom d'attribut en commun, les noms d'attributs de la vue seront obligatoires.

Si, par contre, les noms de colonnes de la vue ne sont pas précisés, celle-ci hérite de ceux des tables sous-jacentes.

Ainsi, on peut définir de la façon suivante la vue V_FROM-BXL qui donne la liste des vols partant de Bruxelles :

```
CREATE VIEW v_from_bxl
        AS SELECT * FROM vols
                WHERE origine = 'BRU ';
```

Une vue constitue donc une “fenêtre dynamique” sur la base de données, dans la mesure où toute modification réalisée sur les données de la vue se reflète automatiquement sur la ou les sources.

Explicitons par un exemple de requête. Cherchons les vols partant de Bruxelles et arrivant à Boston :

```
SELECT * FROM v_from_bxl

        WHERE destin =  'BOS';
```

Lors de l’exécution de cette instruction, celle-ci est d’abord transformée en :

```
SELECT * FROM vols
        WHERE origine = 'BRU' AND destin = 'BOS';
```

S’il s’avère que VOLS est une table et non une vue, la nouvelle instruction est exécutée pour en tirer les tuples recherchées.

UTILITE DES VUES

Il y a deux raisons qui conduisent à définir des vues :

- Tout d’abord, elles augmentent l’indépendance logique des données : l’utilisateur s’affranchit ainsi davantage de l’organisation physique des données. Seules des modifications structurelles des tables se répercuteront sur les vues (exemple : suppression de colonnes) ;
- Dans de nombreux cas, elles servent de mécanisme de protection de la base de données : l’accès des utilisateurs à la base de données sera limité à l’emploi de certaines vues, les données non représentées dans ces vues restant inaccessibles.

Supposons par exemple que l’employé chargé d’enregistrer les passagers à l’embarquement puisse avoir connaissance de leurs noms et des vols associés, sans cependant connaître les coordonnées de ces passagers (adresse, numéro de téléphone, …). On créera à cette fin la vue V_EMBARQUEMENT définie par :

```
CREATE VIEW v_embarquement
        AS SELECT nom, prenom, no_vol
                FROM passagers, reservations
                        WHERE passagers.no_pass = reservations.no_pass ;
```

On peut également limiter la vision d'un utilisateur à des valeurs globales obtenues par des fonctions statistiques plutôt qu'à des valeurs individuelles. Ainsi, une vue peut reprendre le nom de chaque vol et le nombre de places occupées sur ce vol :

```
CREATE VIEW v_charge_vols
        AS SELECT no_vol, COUNT(*) FROM vols
                GROUP BY no_vol ;
```

MISE A JOUR D'UNE VUE

Nous avons vu quelques opérations d'interrogation de vues. Elles ne posent pas de problème majeur, ce qui n'est pas le cas pour la mise à jour des vues. En effet, comme les vues n'ont pas d'existence physique, mais représentent une vision partielle des tables utilisées, il sera nécessaire de prendre des précautions.

Il faudra notamment prendre garde à la répercussion de la mise à jour des attributs de la vue sur les attributs des tables correspondantes. Ces précautions sont davantage des restrictions au niveau de la définition des vues plutôt qu'au niveau de la formulation des instructions de mise à jour.

Par exemple, si l'on a sélectionné certaines colonnes d'une table pour créer une vue, l'ajout d'un tuple à cette vue laissera non définie la valeur des autres colonnes de la table en question, et en particulier la valeur des colonnes impérativement NOT NULL.

Ce problème de la propagation des mises à jour des vues a déjà soulevé de nombreuses polémiques et il n'est pas résolu à ce jour. La solution adoptée par la majorité des SGBD relationnels existants est de limiter fortement les cas de vues pour lesquelles des mises à jour sont permises.

Ainsi, une mise à jour au travers d'une vue V sera autorisée uniquement dans les conditions suivantes :

- La vue V doit être obtenue à partir d'une table unique, ce qui exclut les jointures ;
- Une ligne distincte de la vue V doit correspondre à une ligne distincte de la table ;
- Une colonne distincte de la vue V doit correspondre à une colonne distincte de la table.

Il faut donc que les seules différences entre la table et la vue soient de simples suppressions de ligne ou de colonnes.

Plus concrètement, il découle de ces conditions les restrictions suivantes :

- Si une vue doit faire l'objet de mises à jour (insertions, suppressions, modifications), sa définition ne peut contenir ni d'opérations de jointure, ni de fonctions de calcul, ni de clauses GROUP BY et DISTINCT ;
- De manière générale, le SELECT qui intervient dans la définition d'une vue qu'elle soit, ne peut contenir les clauses ORDER BY et UNION ;
- Si une vue est définie par SELECT comprenant la clause GROUP BY, on ne peut s'en servir dans une jointure, dans une autre table ou une autre vue.

Ainsi, la vue EMBARQUEMENT ne peut servir à l'ajout d'un passager ;

INSERT INTO embarquement(nom, prenom, no_vol)

VALUES ('vandamme','Roger','SN212'); conduira à l'affichage

du message :

… ERROR : #1393 - Can not modify more than one base table through a join view …

SUPPRESSION D'UNE VUE

Une vue sera éliminée par la commande :

```
DROP VIEW nom_vue ;
```

Lors de cet effacement, les vues dont la définition dépend partiellement ou totalement de cette vue sont également supprimées.

Exemple :

DROP VIEW v_embarquement ;

UTILISATION DES INDEX

Pour chaque attribut ou ensemble d'attributs d'une table, il est possible de définir un index. Celui-ci ouvre un accès plus rapide aux informations lors de la manipulation des données. Lors de la création de l'index, les valeurs de chaque colonne qui le constitue seront organisées dans une table d'index selon un ordre croissant ou décroissant, comme l'illustre la figure ci-dessous.

Par la suite, lorsqu'une requête fera intervenir un nom de colonne associé à l'index, la table ne devra plus être parcourue pour obtenir les informations. Il suffira d'utiliser la table d'index pour déterminer, à l'aide du pointeur qui y est stocké, l'emplacement physique du tuple recherché.

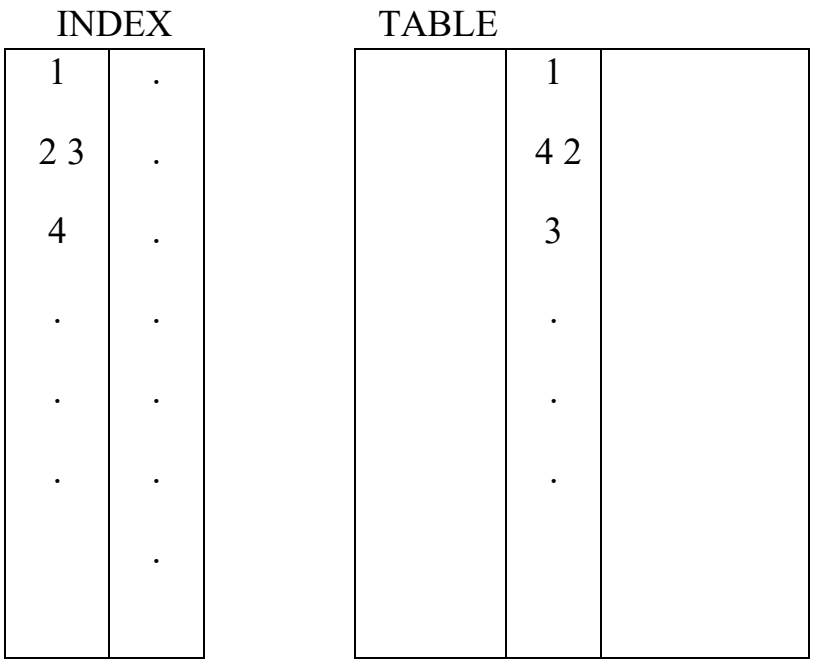

Figure : Le rôle de l'index

On distingue les index simples, relatifs à une seule colonne, des index complexes, relatifs à plusieurs colonnes.

CREATION D'UN INDEX

La commande suivante permet de créer un index :

```
CREATE [ UNIQUE ] INDEX nom_index
                ON nom_table ( nom_col1  [ ASC / DESC ]
[  nom_col 2  [ ASC / DESC ]]) ;
```

On notera qu'un index ne peut recouvrir qu'une seule table, alors que l'on peut créer un nombre quelconque d'index par tables.

L'option UNIQUE de la commande signifie qu'il ne peut y avoir deux tuples dont les valeurs sur la ou les colonnes d'index soient identiques. Si un tel cas se présente lors d'une insertion, l'utilisateur sera averti de l'erreur.

La clause UNIQUE est un facteur de protection de l'intégrité de la base de données, si cette clause s'applique à l'index d'attributs-clés (c'est-à-dire permettant l'accès à un tuple unique lorsqu'ils sont spécifiés).

Pour illustrer ce cas, créons un index sur les noms et prénoms des personnes de la table PASSAGERS :

Exemple :

```
CREATE INDEX  identite
    ON passagers (nom, prenom) ;
```

Remarque :

Il est intéressant de créer des index sur des colonnes souvent prises comme références et pour des tables de grande dimension. Cela permet d'augmenter la vitesse d'exécution des requêtes et des mises à jour.

Cependant, ces index nécessitent une maintenance lors des mises à jour ou des insertions dans la table. Un nombre excessif d'index portant sur une même table conduirait alors à une chute des performances lors de telles opérations.

Il sera donc opportun de dresser l'inventaire des types de questions les plus fréquentes, posées sur la base de données, et ensuite de créer les index les plus appropriés.

Exemple :

Création d'un index sur la table AVIONS en fonction de la marque appareils et de leur autonomie :

```
CREATE INDEX porterparmarque
    ON avions (marque ASC, auto DESC) ;
```

La sélection des avions de portées supérieures à 10000 km au travers de l'index se fera par :

```
SELECT marque, modele FROM avions
    WHERE auto > 10000 ;
```

SUPPRESSION D'UN INDEX

La suppression d'une table d'index suppose l'utilisation de la commande DROP suivant la syntaxe :

```
ALTER TABLE nom_table   DROP  INDEX  nom_index ;
```

Les tables et vues auxquelles l'index est associé ne sont pas affectées par son élimination.

Exemple de suppression d'index :

```
ALTER TABLE avions

    DROP  INDEX  porterparmarque ;
```

En résumé, les vues sont un outil permettant, dans un premier temps, de travailler sur la base de données au travers d'une fenêtre dont le rôle est de nous affranchir des données qui ne nous intéressent pas directement. De plus, dans de multiples cas, elles offrent un moyen efficace d'assurer la sécurité des informations.

Pour accélérer les traitements sur la base de données, les index présentent un intérêt évident à condition de les utiliser rationnellement.