INFORMATIQUE

Laboratoires de bases de données

Laboratoire n°5
Les vues et contraintes

par Danièle BAYERS et Louis SWINNEN
Révision 2024 : Vincent Reip

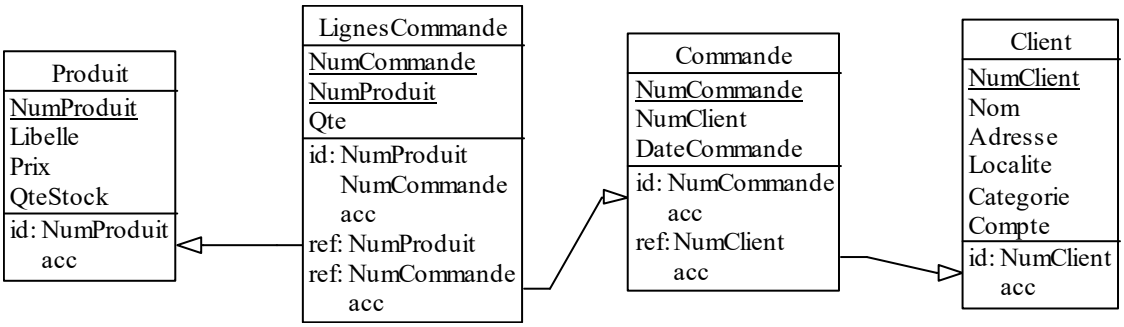
Février 2024

1. Objectif

L'objectif de ce laboratoire est d'aborder la création et l'utilisation des vues ainsi que la mise en place de contraintes simples que l'on peut appliquer à une table.

2. La base de données

Dans les exemples, nous supposerons que nous disposons de la base de données suivante :



3. Les vues

Les vues constituent dans le domaine des bases de données un élément très important. Comme on peut le lire dans [1], la vue va permettre *une facilité d'accès* et *une sécurité accrue des données*. Elles sont également intéressantes dans l'optique *d'évolution de la structure de la base* de données.

Une table, qu'elle soit permanente (contenant les données) ou temporaire (créée pour des raisons statistiques, …), contient des données stables. Ainsi, les données sont ajoutées au moyen de la commande INSERT et modifiées par UPDATE. Une **vue** contient des **données récupérées et traitées dynamiquement** à partir des tables du schéma (voire à partir d'autres vues). La vue n'est pas une table stockée, mais les données qu'elle expose seront générées lorsque l'on interroge cette vue.

Il faut tenir compte cependant du problème suivant : de part sa nature dynamique, la vue peut occasionner un ralentissement au niveau du SGBD si elle est régulièrement utilisée (adapté de [2]).

3.1 Facilité d'accès

Comme précisé dans [1], une vue permet une certaine facilité d'utilisation. En effet, les tables d'une base de données contiennent très souvent des attributs qui ne servent qu'à l'application qui exploite cette base de données (date de la dernière modification, historique des accès, …). Grâce à la vue, il est possible de montrer uniquement les éléments intéressants pour l'utilisateur. Nous avons alors une sorte de table simplifiée. Par extension de ce principe, il peut aussi être intéressant de présenter des données résultant de requêtes complexes (jointures, opérateurs ensemblistes, …). L'utilisateur pourra opérer des requêtes simples sur cette vue sans se soucier de la complexité « cachée » au sein de la définition de la vue.

3.2 Sécurité des données

La vue va également permettre de garantir une certaine sécurité des données. En effet, en donnant accès uniquement aux vues à l'utilisateur, on peut préciser exactement les attributs auxquels ce dernier peut avoir accès. Dans certaines conditions précises, il est possible de modifier des données au moyen de la vue.

3.3 Evolution de la structure de la base

Grâce aux vues, il est possible de faire évoluer la structure de la base de données en toute sécurité par rapport aux applications qui l'exploitent. En effet, la modification de la structure peut apparaître comme transparente pour autant que les vues utilisées restent stables. Les vues opèrent alors comme un niveau d'abstraction « au-dessus » (niveau externe) des tables (niveau interne).

3.4 Les vues en SQL-2

3.4.1 Création d'une vue

```
CREATE VIEW nom_vue (attrv1, …, attrvn)
   AS SELECT attr1, … attrn
      FROM table1
      [JOIN …]
      WHERE condition
      [GROUP BY …]
      [HAVING …]
      [ORDER BY …]
   [WITH READ ONLY | WITH CHECK OPTION]
```

Cette commande permet de créer une vue sur la base de donnée. A chaque « appel » à cette vue, la requête interne sera exécutée et les résultats seront retournés. Il est important de noter que la vue et la requête interne doivent comporter un même nombre d'attribut (#attrv = #attr).

Notons également que la liste d'attributs de la vue ($attrv_1$ … $attrv_n$) n'est pas obligatoire, par défaut, le SGBD reprendra les noms des attributs de la requête interne.

Exemple :

```
CREATE VIEW client_c1 (nom, adresse, localite, compte)
  AS SELECT nomClient, adresse, localite, compte
     FROM client
     WHERE categorie = 'C1'
```

Dans cet exemple, la vue *client_c1* regroupe tous les clients appartenant à la catégorie C1. Il s'agit donc d'une représentation limitée de la table *client* (qui serait par exemple utilisée par le vendeur attitré à ce type de client). L'option WITH READ ONLY / WITH CHECK OPTION sera étudiée ultérieurement.

3.4.2 Destruction de la vue

Tout comme la table, il est possible de détruire une vue. Il faut noter que détruire une vue **ne détruit pas** les données sur lesquelles elle porte.

Format :

```
DROP VIEW nom_vue [RESTRICT | CASCADE]
```

L'option RESTRICT mentionne que la destruction de la vue sera rejetée si elle est utilisée par une autre vue. L'option CASCADE entraîne la suppression de cette vue **ainsi que toutes les autres vues qui dépendraient de celle-ci**.

Exemple :
```
DROP VIEW client_c1 RESTRICT
```

3.4.3 Utilisation d'une vue

Une vue s'utilise comme une table. C'est pourquoi nous allons la retrouver dans l'expression SELECT habituelle.

Format :
```
SELECT attrv1, …, attrvn
FROM nom_vue
WHERE attrvi = val
```

Le SGBD procède alors à l'exécution suivante (extrait de [1]) :
```
SELECT nom_vue.attrv1, …, nom_vue.attrvn
FROM ( SELECT nomClient, adresse, localite, compte
       FROM client
       WHERE categorie = 'C1') AS nom_vue
WHERE nom_vue.attrvi = val
```

3.4.4 Vue modifiable

Pour qu'une vue soit modifiable, il faut respecter les conditions suivantes (extrait de [1]) : «

1. *L'expression de table associée à la vue doit être un simple SELECT, elle ne peut donc contenir les termes JOIN, INTERSECT, UNION ou EXCEPT ;*
2. *La clause FROM ne peut contenir qu'une seule table de base ou une vue elle-même modifiable ;*
3. *L'expression SELECT ne peut contenir la clause DISTINCT ;*
4. *La liste des colonnes du SELECT ne peut comporter d'expression ;*
5. *Si le SELECT contient une requête imbriquée, celle-ci ne peut faire référence à la même table que la requête externe ;*
6. *La requête SELECT ne peut contenir ni GROUP BY, ni HAVING. »*

Si une de ces conditions n'est pas remplie, la vue n'est pas modifiable (impossibilité d'utiliser les commandes INSERT INTO, DELETE ou UPDATE).

Lorsque l'on crée des vues modifiables, il est très souvent utile d'utiliser la clause WITH CHECK OPTION. Grâce à cette clause, le SGBD s'assurera que la modification concerne la vue (impossibilité d'ajouter, de modifier, de supprimer des enregistrements qui n'apparaîtraient pas dans la vue elle-même).

Si on veut empêcher toute modification (INSERT/UPDATE/DELETE) au travers de la vue, on peut utiliser la clause WTH READ ONLY.

Exemple (extrait de [1]):
```
CREATE VIEW client_paris
AS SELECT numClient, nom, adresse, localite, categorie, compte
   FROM client
```

```
   WHERE UPPER(localite) = 'PARIS'
WITH CHECK OPTION
```

Cet exemple crée une vue (CLIENT_PARIS) modifiable montrant les clients de Paris. Il sera donc possible d'ajouter, modifier ou supprimer des tuples de la table sous-jacente (CLIENT) pour autant que la valeur de leur attribut localité soit la chaîne de caractère Paris (peu importe la casse).

4. Les contraintes simples

En plus des contraintes d'identifiant (i.e. les clés primaires) ou encore d'intégrité référentielle (i.e. les clés étrangères), il arrive très souvent que d'autres contraintes doivent être posées sur les données (ex. le prix d'un produit doit être positif, la note obtenue pour une UE doit être comprise entre 0 et 20…). C'est toujours une bonne pratique de d'exprimer les contraintes au niveau du SGBD plutôt que de vérifier ces contraintes par le logiciel. En effet, peu importe la méthode d'accès au SGBD[1], les données de ce dernier restent cohérentes.

Il existe des contraintes d'intégrité facilement identifiable au sein d'un schéma conceptuel. En effet, tout cycle existant dans le schéma conceptuel est très souvent synonyme d'une contrainte d'intégrité. Une grande majorité de ces contraintes est vérifiée au moyen de programmation attachée à la base de données, *les triggers (ou déclencheurs).* Ceux-ci feront l'objet d'une description en profondeur lors d'une prochaine séance de laboratoire.

L'autre type de contrainte d'intégrité concerne un attribut d'une table. Nous avons déjà vu qu'il est possible d'exprimer ces contraintes au moyen de la commande `CHECK`. Nous allons maintenant décrire plus précisément cette commande.

Formats possibles :

```
CHECK ( attrx [>|>=|<|<=|=|BETWEEN|IN val)
CHECK ( attrx [>|>=|<|<=|=|BETWEEN|IN attry)
CHECK ( (SELECT attrx
              FROM table1 t1
              WHERE … ) [>|>=|<|<=|=|BETWEEN|IN] val)
CHECK ( (SELECT attrx
              FROM table1 t1
              WHERE … ) [>|>=|<|<=|=|BETWEEN|IN] attry)
```

Rem : l'utilisation de sous-requête dans une contrainte n'est pas autorisée par ORACLE

Toutes ces contraintes sont attachées à une table. Une fois la contrainte mise en place, toutes les demandes d'insertion au moyen d'INSERT INTO doivent vérifier cette contrainte. De cette manière, il n'est pas possible d'obliger le SGBD à vérifier les données déjà existantes. Cependant, Oracle réalise cette vérification automatiquement et ne vous laissera pas ajouter une contrainte si les données déjà présentes dans le SGBD ne vérifient pas cette nouvelle contrainte.

Pour avoir des contraintes vérifiées à tout moment, il faudrait utiliser les assertions. SQL-2 prévoit la possibilité de déclarer des assertions. Une assertion n'étant pas liée à une table, elle peut contenir une vérification utilisant des jointures. Malheureusement, les assertions ne sont pas implémentées dans les SGBD actuels. C'est la raison pour laquelle nous ne les étudierons pas dans le cadre de ce laboratoire.

---

[1] Pour rappel, les données d'une base de données peuvent être accédées par plusieurs logiciels (y compris les logiciels d'accès au BD tels que SQLDeveloper ou Datagrip).

Exemple :

```
ALTER TABLE produit
 ADD CONSTRAINT prix_positif
  CHECK (prix > 0)
```

Cette requête ajoute une contrainte à la table produit qui vérifie que le prix d'un produit est toujours positif.

```
ALTER TABLE client
 ADD CONSTRAINT categories_client
  CHECK (categorie IN ('C1', 'C2', 'C3', 'C4')
```

Cette requête ajoute une contrainte à la table client qui vérifie que la valeur de l'attribut categorie est C1, C2, C3 ou C4. Toute autre valeur sera rejetée par le SGBD.

Bibliographie

[1] C. MAREE et G. LEDANT, *SQL-2 : Initiation, Programmation*, 2ème édition, Armand Colin, 1994, Paris

[2] P. DELMAL, *SQL2 – SQL3 : application à Oracle*, 3ème édition, De Boeck Université, 2001, Bruxelles

[3] Microsoft, MSDN Microsoft Developper Network, http://msdn.microsoft.com, consulté en février 2009, Microsoft Corp.

[4] Diana Lorentz, et al., Oracle Database SQL Reference, 10g Release 2 (10.2), published by Oracle and available at http://www.oracle.com/pls/db102/homepage, 2005

[5] JL. HAINAUT, *Bases de données : concepts, utilisation et développement,* Dunod, 2009, Paris.

Remerciements

Un merci particulier à mes collègues Vincent REIP et Vincent WILMET pour leur relecture attentive et leurs propositions de correction et d'amélioration.