Chapitre 11

Protection des données

Pour garantir un minimum de protection, tout SGBDR se doit de ne permettre les actions qu'à des utilisateurs parfaitement définis. La politique générale de protection est basée sur l'interaction de trois classes :

les utilisateurs permettent d'identifier les agents qui créent ou modifient les données. Notons d'emblée que l'utilisateur peut, dans bien des cas, être un processus déclenché automatiquement.

les objets de la base de données sont, dans une première définition, les tables. Cependant, dans les systèmes les plus récents, un grand nombre de nouveaux objets sont intégrés dans le système de sécurité.

les privilèges sont accordés à des utilisateurs à propos d'objets.

11.1 Création des comptes, des utilisateurs et des rôles

Les normes SQL ne précisent pas comment créer des utilisateurs. Par contre, elles présupposent leur existence. Les remarques qui suivent ont donc une portée générale et présente des caractéristiques qui peuvent faire défaut sur un système particulier.

- chaque utilisateur est identifié par un nom d'utilisateur unique (*user-id*), limité à un certain nombre de caractères et conforme à la syntaxe des noms de SQL. Oracle utilise le mot réservé `USER` pour renvoyer le nom de l'utilisateur de la session courante[1]. Sur la plupart des systèmes, le nom de l'utilisateur se double d'un numéro unique également (dans ce cas, les permissions sont gérées sur base de ce numéro, ce qui a pour conséquence qu'un utilisateur effacé puis réintroduit ne retrouve aucun de ses privilèges). Oracle utilise le mot réservé `UID` pour obtenir ce numéro unique.
- l'authentification de l'utilisateur se fait à travers un mot de passe. Selon le cas, le mot de passe sera demandé à chaque nouvelle session ou au contraire mémorisé dans le code d'un programme. Cette dernière possibilité, qui permet à un programme lancé automatiquement à un moment donné de réaliser une tâche sans intervention d'un utilisateur humain, doit faire l'objet d'une attention particulière, parce que quelqu'un peut essayer d'utiliser le programme pour jouir de ses droits.
- comme souvent plusieurs humains doivent disposer des mêmes droits parce qu'ils effectuent un même travail, on doit trouver un moyen de gérer ces droits identiques. Trois solutions sont possibles :

[1] Pour l'afficher, on peut utiliser la pseudo table `Dual` avec la commande suivante :

```
SELECT USER FROM Dual
```

©2008-2009 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

- permettre à plusieurs personnes de se connecter sous un même nom d'utilisateur (plusieurs sessions simultanées). Dans ce cas, on perd la possibilité de les distinguer. La gestion des mots de passe est également très lourde puisque pour interdire l'accès d'un utilisateur physique on est obligé de changer le mot de passe utilisé par tous.
- accorder les privilèges sur une base individuelle. Outre le caractère fastidieux du travail, qui peut cependant être allégé par le recours à des scripts, cette méthode rend très lourde la moindre modification des privilèges.
- certains systèmes permettent la création de groupes, nommés *rôles*, ce qui réduit considérablement le travail puisque les autorisations sont accordées aux rôles et qu'il suffit alors d'inscrire les utilisateurs dans le ou les rôles qui correspondent à leurs besoins du moment.

Nous allons explorer la solution proposée par Oracle, qui ressemble bien entendu à toutes les solutions proposées par les SGBD professionnels. J'ai choisi de n'aborder la gestion des utilisateurs qu'à travers l'usage de la console. On peut bien entendu gérer tout cela à l'aide de l'interface graphique, d'une manière très intuitive (ne pas oublier le clic droit). Il faut noter cependant que l'utilisation de scripts permet de planifier les tâches, d'en garder une trace à des fins de documentation ou de réparation. **Ces scripts devraient être mis dans un endroit sûr (coffre-fort ou fichier crypté) parce qu'ils contiennent en clair certains mots de passe[2].** Ils constituent également une arme de choix, même sans mot de passe, pour un attaquant. L'ignorance des noms d'utilisateurs constitue une barrière supplémentaire.

11.1.1 Utilisateurs d'une base de données Oracle

Par défaut, Oracle dispose de deux utilisateurs prédéfinis qui sont créés en même temps que la base de données. Ce sont *sys*, propriétaire du dictionnaire des données, et *system*. Les premières connexions se font sous le nom de *system*, qui créera les premiers utilisateurs réels. Il faut donner à cet administrateur un mot de passe sérieux et ne pas le perdre, car la gestion de la base de données dépend entièrement de lui[3].

La création d'un utilisateur nécessite des droits dont seuls les administrateurs disposent par défaut.

---

[2] Le choix d'un mot de passe devrait se faire avec réflexion. Quelques mots de passe à éviter :
- un simple nombre ;
- le nom d'une personne chère ou, pire, son propre nom ;
- un nom à l'envers (`snerooht`)
- un mot d'une langue européenne ;
- une séquence du clavier (`azert`) ;
- un personnage historique ou du monde informatique (`Codd`).

Voici quelques exemples de bons mots de passe :
- une combinaison de lettres majuscules, minuscules et de signes de ponctuation (ex `uf,P4$y`)
- les initiales d'une phrase n'ayant aucun rapport ni avec l'utilisateur, ni avec l'informatique (La Marquise Sortit A Cinq Heures -> `lmsach`). On peut corser en alternant majuscules et minuscules (`LmSaCh`) ou encore remplacer les voyelles par leur position (aeiou/12345 -> `LmS1Ch`). Ce système, fonctionnant avec une phrase simple, a l'avantage d'être plus facile à mémoriser.

Enfin dernier point, un mot de passe (ou la combinaison de votre carte de banque) ne doit jamais être écrit nulle part, surtout pas en dessous du clavier, ni dans un carnet d'adresse. Un pirate sera intrigué par vos relations avec M. LmSach ou par le numéro de téléphone 0381018.

[3] Sous Linux, il reste possible de se connecter comme administrateur sans mot de passe à condition d'avoir ouvert une console comme utilisateur *oracle*, qui est l'utilisateur Linux qui a installé et possède les fichiers constituant le serveur. L'administrateur *root* de la machine hébergeant le serveur peut donc, en cas de catastrophe, usurper l'identité d'*oracle* pour réparer le mot de passe perdu.

©2008-2009 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

CREATE USER *Nom_de_Compte*
IDENTIFIED BY *"motdepasse";*

Chaque utilisateur possède un profil, qui spécifie les conditions de connexion, de travail et de modification du mot de passe. Sauf spécification contraire, un nouvel utilisateur est inscrit dans le profil `DEFAULT`, dont voici la définition ;

```
CREATE PROFILE "DEFAULT"
LIMIT CPU_PER_SESSION UNLIMITED CPU_PER_CALL UNLIMITED
CONNECT_TIME UNLIMITED IDLE_TIME UNLIMITED
SESSIONS_PER_USER UNLIMITED LOGICAL_READS_PER_SESSION UNLIMITED
LOGICAL_READS_PER_CALL UNLIMITED PRIVATE_SGA UNLIMITED
COMPOSITE_LIMIT UNLIMITED FAILED_LOGIN_ATTEMPTS UNLIMITED
PASSWORD_LOCK_TIME UNLIMITED PASSWORD_GRACE_TIME UNLIMITED
PASSWORD_LIFE_TIME UNLIMITED PASSWORD_REUSE_MAX UNLIMITED
PASSWORD_REUSE_TIME UNLIMITED PASSWORD_VERIFY_FUNCTION NULL
```

Si le besoin s'en fait sentir, on peut créer d'autres profils et y attribuer des utilisateurs, au moment de la création ou plus tard.

```
ALTER USER util01 PROFILE mon_profil;
```

Si pour une raison quelconque, on veut temporairement interdire la connexion d'un utilisateur ou la réactiver, on peut utiliser la commande `ACCOUNT (UN)LOCK`. Voici la commande complète pour la création d'un utilisateur, pour lequel on précise les espaces où il pourra créer éventuellement sa base de données (qui fait l'objet d'un droit supplémentaire) et les quota autorisés. Sans quota, la création de tables n'est pas possible.

```
CREATE USER "UTIL01"  PROFILE "MON_PROFIL"
IDENTIFIED BY "********" DEFAULT TABLESPACE "USERS"
QUOTA 200 M ON "USERS"
TEMPORARY TABLESPACE "TEMP"
QUOTA UNLIMITED ON "TEMP"
ACCOUNT UNLOCK
```

Les commandes `ALTER USER` et `DROP USER` permettent respectivement de modifier et de supprimer un utilisateur.

11.1.2 Utilisateurs, bases de données et schémas

On a vu qu'un serveur Oracle ne comportait généralement qu'une seule base de données. Les unités logiques correspondant à des besoins précis sont constituées par un schéma. Un schéma porte le nom de son créateur. Nous allons créer un utilisateur *Rochefort* qui créera son schéma.

```
CREATE USER Rochefort IDENTIFIED BY "1234"
DEFAULT TABLESPACE "USERS"
QUOTA UNLIMITED ON "USERS"
TEMPORARY TABLESPACE "TEMP"
QUOTA UNLIMITED ON "TEMP";
-- autorisation de se connecter
GRANT CONNECT TO Rochefort;
```

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

```
-- autorisation de créer des tables (donc le schéma)
GRANT CREATE TABLE TO Rochefort;
```

Quand *Rochefort* créera sa première table, il va implicitement créer un nouveau schéma portant son nom. Il reste maintenant à l'administrateur ou à Rochefort lui-même à autoriser un autre utilisateur à utiliser les tables de ce schéma[4].

11.1.3 Modification des mots de passe

Un mot de passe est créé en même temps que l'utilisateur (au moyen de la clause `IDENTIFIED BY "mdp"`) par le créateur de l'utilisateur. L'utilisateur peut modifier son mot de passe par la commande suivante :

```
ALTER USER nomutilisateur IDENTIFIED BY "mdp";
```

Pour changer le mot de passe d'un autre utilisateur, il faut être soit *system* ou en avoir reçu le privilège :

```
GRANT ALTER USER TO nomutilisateur
```

11.1.4 Rôles dans Oracle

Les rôles permettent de simplifier la gestion des privilèges. On attribue les privilèges à des rôles, qui rassemblent des privilèges communs à plusieurs utilisateurs. Il ne reste plus qu'à attribuer des rôles aux utilisateurs.

Oracle est fourni avec plusieurs rôles prédéfinis. Certains d'entre eux sont des rôles de haut niveau qui correspondent à des responsabilités importantes dans la base de données : gestionnaires de l'archivage, de la restauration, manipulateurs du catalogue des données. Les rôles `CONNECT` et `RESOURCE` ont joué un rôle important dans les versions antérieures d'Oracle. Ils ne seront plus supportés dans les versions ultérieures. Le rôle `DBA` permet de créer des administrateurs à peu de frais. La figure 11.1 montre que le rôle est en fait obtenu par la combinaison de plusieurs rôles secondaires, imbriqués. Cette interface n'est pas accessible dans SQL-Developper.

L'administrateur, et ceux à qui il transmet ses pouvoirs, sont en mesure de créer des rôles. Un rôle est normalement défini sans mot de passe par la commande suivante :

```
CREATE ROLE NomRole NOT IDENTIFIED;
```

On peut aussi lui attribuer un mot de passe, qui impose de le taper quand on l'utilise. Les commandes `ALTER ROLE` et `DROP ROLE` permettent de modifier et de supprimer un rôle.

Un rôle se gère comme un privilège (voir plus loin). Les deux commandes suivantes attribue et retire respectivement un rôle à l'utilisateur *dupont*.

```
GRANT Bidouilleur TO dupont;
REVOKE Bidouilleur FROM dupont;
```

[4] On peut même sauter une étape en attribuant le droit de connexion à un utilisateur sans l'avoir préalablement créé :

```
GRANT CONNECT TO Rochefort IDENTIFIED BY "1234";
```

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

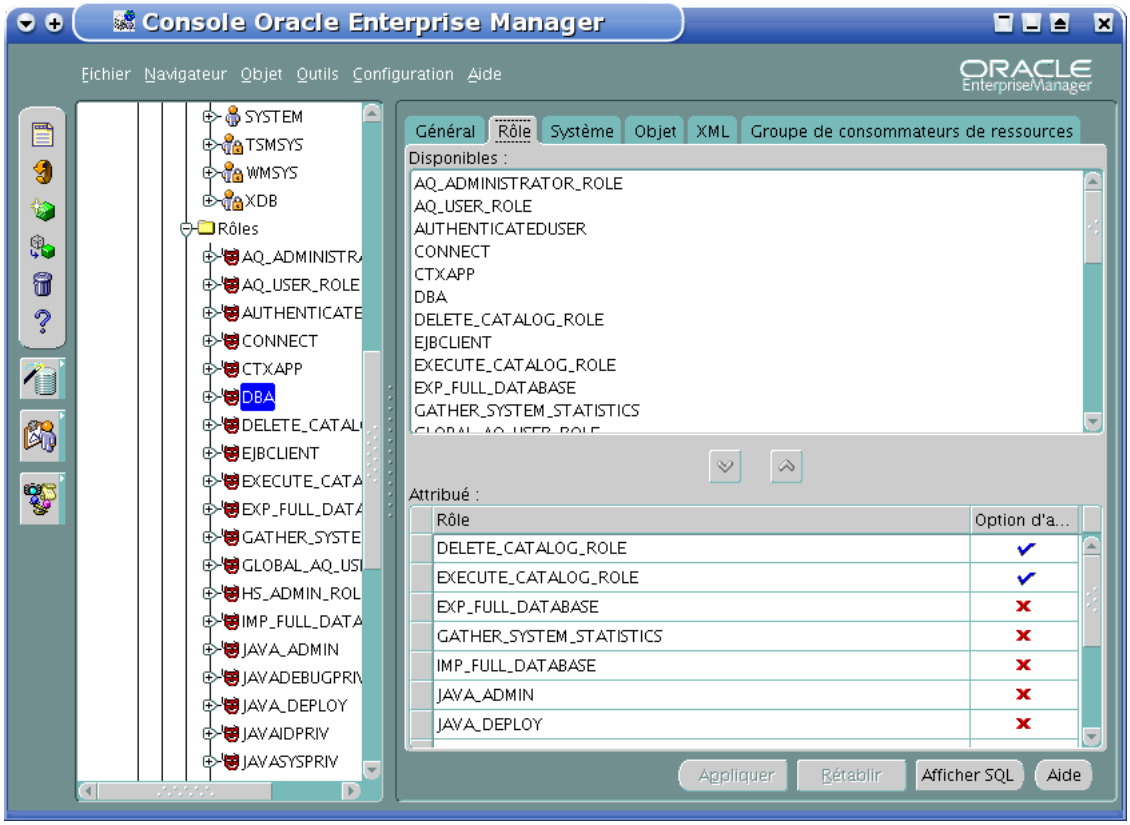

FIG. 11.1 – Interface de manipulation des rôles

Si un utilisateur possède plusieurs rôles, il cumule les privilèges de chacun de ses rôles. Il peut néanmoins activer et désactiver ses rôles.

```
CREATE USER dupont IDENTIFIED BY "1234";
CREATE ROLE bidouilleur NOT IDENTIFIED;
CREATE ROLE aspirateur NOT IDENTIFIED;
CREATE ROLE nettoyeur IDENTIFIED BY "ajax";
GRANT bidouilleur,aspirateur,nettoyeur TO dupont;
GRANT CONNECT TO dupont;
CONNECT DUPONT/1234@eci2;
SET ROLE none;
SET ROLE bidouilleur;
...
SET ROLE nettoyeur identified by "ajax";
...
SET ROLE ALL;
```

Cette dernière instruction échouera parce que Dupont aurait dû fournir le mot de passe pour récupérer le rôle de *nettoyeur*. Par contre, il avait obtenu ce rôle par défaut (et sans mot de passe) lors de sa connexion.

Un problème se pose avec les rôles : ils ne peuvent posséder aucun objet. Cela impose donc aux utilisateurs faisant partie d'un rôle d'utiliser le schéma d'un autre utilisateur (avec la nécessité de préfixer chaque nom d'objet par son nom). Cela conduit certains programmeurs à n'utiliser qu'un seul utilisateur pour un programme : le propriétaire des objets manipulés. On peut aussi recourir aux synonymes, mais on doit forcément utiliser des synonymes publics, ce qui risque d'entraîner des conflits de noms entre les différents schémas.

©2008-2009 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

11.1.5 Variables et fonctions d'environnement

Il est souvent nécessaire de vérifier quel utilisateur est connecté (et quelques renseignements le concernant). Oracle possède deux variables `UID` et `USER` qui renvoient le numéro et le nom de l'utilisateur connecté. On peut également utiliser la fonction `USERENV` et l'un de ses paramètres pour obtenir des renseignements sur la session en cours.

paramètre	utilisation
`SESSIONID`	numéro de la session
`TERMINAL`	nom du terminal sur le système exploitation hôte (renvoie *null* sur iSQL*Plus)
`ENTRYID`	numéro de la requête dans la session en cours
`LANGUAGE`	la langue de l'utilisateur (une session en français renvoie la valeur suivante : FRENCH_FRANCE.WE8ISO8859P1)

`USERENV` est en cours de remplacement par `sys_context()`, qui reprend ses options et en ajoute d'autres, mais la syntaxe est plus lourde :

```
select userenv('language') from dual;
select sys_context('userenv','language') from dual;
```

11.2 Gestion des privilèges

Il existe deux types de privilèges :

– les *privilèges système* précise des droits sur des comportements (se connecter, créer des tables, ajouter des utilisateurs...). Ils se définissent au niveau de la base de données et ne souffrent pas d'exception : si quelqu'un a le droit de supprimer des utilisateurs, cela concerne tous les utilisateurs de la base de données. Au départ, personne, à part l'administrateur, ne dispose du moindre privilège système.

– les privilèges sur les objets s'appliquent à des objets précis, qui ont en outre un propriétaire. Seuls les administrateurs et les propriétaires des objets possèdent le droit de manipuler les objets. Ils peuvent néanmoins donner des privilèges à d'autres utilisateurs ou à des rôles sur les objets qu'ils manipulent.

Il existe un moyen pour l'administrateur de se décharger sur d'autres de son travail : la clause « `with admin option` » ajoute au privilège le droit de le transmettre à d'autres. Un administrateur pourrait par exemple créer un adjoint spécialement chargé de la création des utilisateurs.

```
GRANT create user TO UserCreator IDENTIFIED BY "1234"
WITH ADMIN OPTION;
GRANT create session TO UserCreator;
```

Cet adjoint pourra créer tous les utilisateurs qu'il veut et même donner à qui il voudra le droit de créer d'autres utilisateurs.

Oracle ne retient pas l'origine d'un privilège (à la différence de SQL-Server). Si on retire un privilège à quelqu'un qui l'a transmis à d'autres, ceux qui ont obtenu ce privilège le gardent[5]. Si

[5]Dans le système de Microsoft, la suppression d'un droit à quelqu'un entraîne la suppression du même droit à tous ceux qui l'ont obtenu de lui. Cette gestion des héritages mène à des situations conflictuelles puisqu'une personne peut obtenir un même droit de deux autres personnes. En cas de révocation, on peut arriver à des situations peu claires pour l'administrateur.

©2008-2009 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

on veut retirer en cascade, cela implique du travail (examiner tous les utilisateurs qui disposent du privilège concerné). Il n'est pas certain cependant que le retrait d'un privilège, motivé par un départ, un changement d'affectation ou un abus de pouvoir, concerne les subalternes de l'utilisateur. La conception d'Oracle se défend parfaitement. C'est ici qu'une gestion des privilèges sur base des rôles se justifiera pleinement en apportant une réelle simplification.

On utilise les commandes `GRANT` et `REVOKE` pour accorder et retirer les privilèges.

```
-- accorder un privilège
GRANT CREATE USER TO Utilisateur1;
-- accorder plusieurs privilèges
GRANT CREATE USER, CREATE ROLE TO Utilisateur1;
-- accorder tous les privilèges
GRANT ALL PRIVILEGES TO Utilisateur1;
-- accorder un privilège et possibilité de le transmettre
GRANT CREATE USER TO Utilisateur1 WITH ADMIN OPTION;
-- accorder un privilège et créer un utilisateur
GRANT CREATE CONNECTION TO Nouveau IDENTIFIED BY "1234";
-- retirer plusieurs privilèges
REVOKE CREATE USER, CREATE ROLE FROM Utilisateur1;
-- retirer tous les privilèges
REVOKE ALL PRIVILEGES FROM Utilisateur1;
```

Cette dernière commande n'est pas un raccourci pour paresseux. Elle retire tous les privilèges à celui qui les auraient reçus tous par la commande GRANT correspondante. Elle échoue si on l'applique à un utilisateur qui ne dispose que de quelques privilèges.

SQL Server propose une variante intéressante de `GRANT`, `DENY`, qui a l'effet exactement opposé. La syntaxe est identique à celle de `GRANT`. Son utilité est de pouvoir, par exemple, interdire une action à une personne appartenant à un rôle qui est normalement autorisé à réaliser cette action.

11.2.1 Privilèges système

Il n'est pas possible d'énumérer tous les privilèges système. On pourra en voir une liste exhaustive dans l'interface de la console d'administration. Le tableau 11.1 reprend les principaux privilèges.

On notera que l'absence de `ALTER` et `DROP` à propos de `INDEX`, `TABLE` et `SEQUENCE` est logique. En créant ces types d'objets, l'utilisateur en devient propriétaire et dispose donc sur eux de droits absolus. Les privilèges comportant `ANY` portent sur des objets qui n'appartiennent pas à l'utilisateur. Les commandes suivantes donnent à Arthur le droit de créer son schéma et d'y définir des tables, à Bernard de créer une table où il veut et retire à Charles le droit de créer des séquences :

```
GRANT CREATE TABLE TO Arthur
GRANT CREATE ANY TABLE TO Bernard
REVOKE CREATE SEQUENCE FROM Charles
```

11.2.2 Privilèges sur les objets

SQL1 : privilèges de base

La première norme SQL prévoyait des privilèges sur les objets de base (voir tableau 11.2).

©2008-2009 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

Privilège	ALTER	CREATE	DROP
INDEX		x	
ANY INDEX	x	x	
TABLE		x	
ANY TABLE	x	x	x
USER/ROLE	x	x	x
PROFILE	x	x	x
SEQUENCE		x	
ANY SEQUENCE	x	x	x

TAB. 11.1 – Privilèges systèmes

Privilèges	**Table**	**Vue**	**Séquence**	**Programme**
ALTER			x	
DELETE	x	x		
EXECUTE				x
INDEX				
INSERT	x	x		
REFERENCES	x			
SELECT	x	x	x	
UPDATE	x	x		

TAB. 11.2 – Privilège sur les principaux objets

SQL2 : colonnes

Le standard SQL2 autorise la spécification des droits au niveau des colonnes (du moins pour les actions `INSERT`, `UPDATE` et `REFERENCES`). Il suffit de préciser entre des parenthèses les champs qui sont concernés par le privilège.

```
GRANT UPDATE (Prix) ON Modeles TO Marcel
```

En principe, la norme n'autorise les privilèges sur les champs que pour les trois actions précitées. Il est vrai qu'on peut facilement autoriser des accès en lecture plus fins au moyen d'une vue, couplée à son tour à la distribution de privilèges (voir section suivante). En pratique, beaucoup de SGBDR ont étendu la norme à l'action `SELECT`. C'est le cas d'Oracle. Bien que SQL Server 7.0 autorise la définition des droits sur les colonnes, l'interface graphique n'offre plus de moyen de le faire.

SQL2 : `REFERENCES`

Le privilège `REFERENCES` autorise un utilisateur à utiliser une clé étrangère qui pointe sur une clé primaire d'une table à laquelle il n'a pas accès. Sans cette permission, il faudrait lui donner un droit sur l'action `SELECT` (éventuellement limitée à la clé primaire, si le système le permet). Dans Oracle, ce privilège ne peut s'accorder qu'à des utilisateurs et non à des rôles.

Les exemples suivants définissent une série de droits sur des tables pour des groupes d'utilisateurs. Il peut y avoir plusieurs droits et plusieurs utilisateurs/groupes, mais les commandes portent sur un seul objet à la fois.

```
GRANT SELECT ON Articles TO Vendeurs
GRANT INSERT,SELECT,UDATE,DELETE
ON Articles TO Direction, Facturation
```

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

11.3 Utilisation des vues

Les vues sont des tables virtuelles qui se basent sur des données placées dans des tables réelles ou d'autres vues. Elles sont toujours synchronisées avec les données réelles (ce ne sont donc pas des copies temporaires, parfois appelées *snapshots*). Elles ne sont pas toujours modifiables, mais quand elles le sont, parfois partiellement, elles entraînent la modification des tables réelles. Si la vue n'a d'autre fonction que de fournir un autre nom à une table, Oracle propose des *synonymes*, objet de la section suivante.

11.3.1 Leur utilité

Les vues répondent à plusieurs besoins dans une base de données :
- elles servent d'interface entre la vue logique (celle des tables) et la vue externe, celle que perçoit l'utilisateur. On peut donc simuler l'existence de tables qui n'existent pas ou qui n'existent plus. C'est un moyen de séparer la structure des données telle que la voient les utilisateurs (et surtout les applications) et la structure réelle de la base de données.
- elles permettent de masquer des données en fonction des utilisateurs. En effectuant des projections et des restrictions, on ne montre à certains utilisateurs que les colonnes et les lignes qui les concernent. Plutôt que de définir des privilèges compliqués sur des tables, on préféra parfois définir plusieurs vues qu'on offrira à différents types d'utilisateurs.
- comme elles sont basées sur une requête `SELECT`, elles permettent d'enregistrer des requêtes complexes que des utilisateurs peu expérimentés seraient bien en peine d'écrire eux-mêmes. Elles dispensent donc de retaper du code compliqué et facilitent la gestion des requêtes utilisées par les programmes en les plaçant sur le serveur.

11.3.2 Définition d'une vue

La définition d'une vue est toujours basée sur les éléments suivants :
- un nom unique (qui ne peut être celui d'une table existante)
- une série d'alias pour les noms de colonnes
- la requête proprement dite.

Assez curieusement, les alias peuvent se définir soit dans l'en-tête de la vue, soit à l'aide d'alias normaux dans la requête. En l'absence des uns et des autres, les colonnes portent le nom de la colonne originale ou un nom généré par le système s'il s'agit de champs calculés.

```
CREATE VIEW octobre.ecole1 as
SELECT nom, interro1+interro2 Semestre, Examen Exa
FROM Octobre.Etudiants;
```

Il est maintenant possible d'utiliser *Ecole1* comme une table.

```
SELECT semestres FROM octobre.ecole1;
```

Voici une variante de la même vue, dont les noms de colonnes ne sont plus des alias mais sont ici définis dans la vue :

```
CREATE VIEW Octobre.Ecole2 (NomEtudiant,Semestres,Exa) AS
SELECT nom,Interro1+Interro2,Examen
FROM Octobre.Etudiants;
```

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

11.3.3 Possibilité de modifier les données d'une vue

La loi du bon sens prévaut dans la possibilité de modifier des données à l'aide d'une vue : si c'est possible en théorie, cela devrait être possible en pratique. Nous distinguerons le cas des vues monotable, multitables et le cas particulier des restrictions. Rappelons avant tout que la modification peut concerner l'insertion, la mise-à-jour et la suppression.

Vues monotables

Il existe des vues qui ne sont systématiquement pas modifiables : celles où une ligne de la vue correspond à plusieurs lignes de la table originale : les vues comportant `DISTINCT`, `GROUP BY` ou une fonction statistique. Pour ce qui concerne les autres, le principe général de bon sens prévaut et elles sont en principe modifiables. Quelques contre-exemples :

- une vue qui ne contient pas un champ déclaré `NOT NULL` ne permettra pas l'insertion ;
- une vue qui ne contient pas une clé étrangère (implicitement non nulle) ne permettra pas l'insertion ;
- une vue qui ne contient pas la clé primaire permettra l'édition, la suppression mais pas l'insertion, la même règle s'applique pour les clés candidates (`UNIQUE`) non présentes ;

Il va de soi que les contraintes normales (intégrité référentielle, validation des contraintes) restent d'application.

Vues multitables

Il est plus difficile de parler de bon sens dans le cas d'une vue multitable et les SGBDR proposent des solutions différentes. InterBase se montre le plus restrictif et ne tolère aucune modification dans ce contexte, ce qui simplifie sans doute le travail de ses concepteurs mais limite trop la créativité des utilisateurs. MS-Access est très permissif, mais parfois au dépend de la clarté. On pourra par exemple modifier le nom du client dans une vue affichant des descriptions de factures, mais on modifie en fait le nom du client dans la table client, alors que l'utilisateur entendait peut-être attribuer la facture à un client différent.

Oracle adopte une position médiane et autorise la modification d'une seule table de la vue, selon des conditions précises. Cette possibilité est évidemment soumise aux même conditions préalables qu'une vue monotable. On fait intervenir la notion de « table protégée par sa clé » :

> *une table est dite* protégée par sa clé (key preserved) *si sa clé primaire est préservée dans la clause de jointure et se retrouve en tant que colonne de la vue multitable (elle peut jouer le rôle de clé primaire de la vue).*

C'est le cas de la liste des productions obtenue en joignant une liste de séries télévisées et leurs producteurs. On pourra insérer une nouvelle série, modifier les données provenant de la table *Séries*, y compris la clé étrangère désignant le producteur. Par contre, les données provenant de la table des producteurs sont en simple lecture. En cas d'insertion, les données apparaissant dans la vue dépendront de la clé étrangère.

```
CREATE OR REPLACE VIEW Productions AS
SELECT idSerie,TitreSerie,S.idProducteur, NomProd
FROM Series S INNER JOIN Producteurs P
ON S.idProducteur=P.idProducteur;
```

On peut interroger le système pour obtenir une indication sur les possibilités de modification des colonnes d'une vue :

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

```
SELECT column_name, insertable, updatable, deletable
FROM USER_UPDATABLE_COLUMNS WHERE TABLE_NAME='PRODUCTIONS';

COLUMN_NAME                    INS UPD DEL
------------------------------ --- --- ---
IDSERIE                        YES YES YES
TITRESERIE                     YES YES YES
IDPRODUCTEUR                   YES YES YES
NOMPROD                        NO  NO  NO
```

Vérifications des restrictions `WHERE`

Lorque la requête qui fonde une vue comporte une clause `WHERE`, rien n'empêche *a priori* d'apporter une modification (insertion ou mise-à-jour) qui fait qu'une ligne ne devrait pas être montrée.

```
CREATE VIEW Data.MesDonnees AS
SELECT * FROM Data.DonneesDeTous
WHERE Proprio = UID;
```

L'utilisateur 59 ne verra que les lignes qui correspondent à son numéro d'utilisateur. S'il encode une ligne comprenant un numéro de propriétaire différent du sien, il ne la verra pas lors d'une requête ultérieure à l'aide de la vue.

```
 SQL> select * from donneesdetous;
       CLE TEXTE                PROPRIO
---------- -------------------- ----------
         1 un                           59
         2 deux                         59
         3 trois                        60
         4 quatre                       59
         5 cinq                         61
SQL> insert into mesdonnees values(6,'cinq',62);
1 ligne créée.
SQL> select * from mesdonnees;
       CLE TEXTE                PROPRIO
---------- -------------------- ----------
         1 un                           59
         2 deux                         59
         4 quatre                       59
```

Pour éviter cette situation illogique, Oracle permet de renforcer la clause `WHERE` à l'aide de la directive `WITH CHECK OPTION`. Dans ce cas, l'insertion ou la mise à jour d'une ligne ne répondant plus à la condition devient impossible.

```
CREATE OR REPLACE VIEW MesDonnees AS
SELECT * FROM Octobre.DonneesDeTous
WHERE Proprio = UID WITH CHECK OPTION;
```

Tentons une modification :

```
SQL> update mesdonnees set Proprio=60 where Cle=1;
* ERREUR à la ligne 1 : ORA-01402:
vue WITH CHECK OPTION - violation de clause WHERE
```

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

11.4 Les synonymes

Parfois la seule nécessité de créer une vue vise à disposer d'un nom différent pour une table. La notion de synonyme permet d'y parvenir à moindre frais. Il existe deux sortes de synonymes : les synonymes publics et les synonymes limités à un seul schéma. Les synonymes propres à un schéma sont assez peu intéressants puisqu'ils ne sont accessibles qu'à leur seul propriétaire, mais on peut les redéfinir dans tous les schémas où ils seraient nécessaires à l'aide de scripts. Les synonymes publics permettent notamment d'alléger l'écriture du nom des tables en supprimant la mention du schéma. Il faut penser pourtant que cette solution met tous les noms de table dans un seul espace de noms et, pour de gros projets, les risques de conflits sont assez grands. Voici deux exemples de synonymes :

```
CREATE SYNONYM Invoices FOR Factures;
CREATE PUBLIC SYNONYM Factures FOR Reception.Factures;
```

La dernière commande montre un problème : le service comptabilité devra trouver un autre nom pour sa table des factures.

11.5 Les bases de données privées virtuelles d'Oracle

Les dernières versions d'Oracle offrent un moyen efficace de protection des données par le biais de bases de données privées virtuelles (VPD). Il s'agit d'un ensemble de techniques qui autorisent l'installation de conditions de restrictions sur toutes les tables, vues et synonymes, une sorte de `WHERE` implicite. La mise en œuvre de ces VPD nécessitent l'usage de plusieurs techniques que nous n'avons pas encore abordées. Je me contenterai donc d'en exprimer le principe, vu qu'un chapitre leur sera consacré dans la dernière partie.

1. définition d'un paquetage de procédures lié à un contexte d'application.
2. écriture d'une procédure qui va tester l'utilisateur en cours et créer une variable d'environnement qui permettra de le décrire (un numéro par exemple) et son placement dans le paquetage.
3. création d'un trigger de connexion qui va exécuter la procédure du point précédent lors de toute connexion, ce qui aura pour effet d'initialiser la variable d'environnement en question avec la bonne valeur.
4. création d'une politique de sécurité matérialisée par des fonctions dans un paquetage qui renvoie une valeur vraie ou fausse en fonction de la variable d'environnement définie au point 2
5. liaison des politiques de sécurités aux tables concernées.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET