Chapitre 2 : Gestion d'accès sous ORACLE

2.1/ Droits d'accès

Plusieurs utilisateurs peuvent accéder même table d'une base (en tant que client ou par accès direct).

Les données peuvent être confidentielles ou partagées entre plusieurs utilisateurs

Contrôle de l'accès à la base

- Le contrôle de l'accès à la base est effectué en associant à chaque utilisateur
  un nom de login
  un mot de passe

- Chaque utilisateur a des privilèges d'accès à la base : **droit ou non**
  de créer des tables ou des vues,
  de lire ou de modifier des tables ou des vues

Propriétaire des données

- Une table (et les données qu'elle contient) appartient à celui qui l'a créé

- Le propriétaire d'une table peut donner à d'autres le droit de travailler avec sa table

- Les vues permettent d'affiner les droits que l'on donne sur ses propres données : On peut donner des droits sur des vues et pas sur les tables sous-jacentes.

Intérêt des vues? *Donner des droits à une partie d'une table*

Exemple :

- On considère la table note (nom, matiere, note)

- ```
  CREATE VIEW note_bd (nom, matiere, note)
                  AS select  * from note where matière = 'BD'
  ```

- ```
  Grant update on note_bd to Michel ;
  ```
  Michel est le l'enseignant responsable du cours BD

Accorder des droits

- `GRANT privilège ON table/vue TO utilisateur [`**`WITH GRANT OPTION`**`]`

- Des privilèges :

```
SELECT
INSERT
UPDATE [(col1, col2,…)]
DELETE
INDEX
ALTER
ALL
```

L'option `WITH GRANT OPTION` permet à l'utilisateur qui reçoit le privilège de le donner à d'autres utilisateurs.

Accorder des droits (exemples)

- `grant select on emp to clement;`
- `grant select, update on emp to clement, chatel;`
- `grant select on emp to public;`

Changer son mot de passe :

- `grant connect to bibi  identified by motDePasse;`

Reprendre les droits

- `REVOKE privilège ON table/vue FROM utilisateur`

2.2/ Les transactions

Début d'une transaction

- Dans la norme SQL2 toute modification appartient à une transaction
- Une transaction démarre au début de la session
- Une transaction démarre automatiquement après la fin d'une transaction (pas de commande pour démarrer une transaction en SQL2)
- La structure des transactions est « plate » : les transactions ne peuvent se chevaucher

Terminer une transaction

- Pour terminer une transaction on peut
  - valider la transaction (COMMIT) : toutes les modifications deviennent effectives
  - annuler la transaction (ROLLBACK) : toutes les modifications sont annulées

- Les ordres DDL (create table par exemple) provoquent un COMMIT automatique

Propriétés des transactions

- **A**tomicité : un tout indivisible

- **C**ohérence : une transaction doit laisser la base dans un état cohérent ; elle ne doit pas mettre les données dans un état « anormal »

- **I**solation : les données non validées d'une transaction ne doivent pas être vues par les autres transactions

- **D**urabilité : le SGBD doit garantir que les modifications d'une transaction validée seront conservées, même en cas de panne

Exemple d'annulation d'une transaction

```
insert into dept values (10, 'Finance', 'Paris');
delete from emp;
rollback;
```

Transaction « non terminée »

- La dernière transaction est automatiquement validée à la sortie de SQL*PLUS si elle ne se termine pas par un ROLLBACK

- Ce comportement dépend du logiciel utilisé ; avec d'autres logiciels une transaction non validée explicitement est annulée

Isolation des transactions

En fonctionnement standard les modifications effectuées par une transaction ne sont connues des autres transactions qu'après sa validation

Transactions longues Exemple :

Organisation par une agence de voyage d'un voyage Nice – Wuhan (Chine)
Nécessite la réservation de plusieurs billets d'avion : Nice – Paris ; Paris – Beijing ; Beijing – Wuhan.

On commence par réserver les 2 premiers mais si on ne peut trouver de Beijing – Wuhan, il faut tout annuler.

On met donc toutes ces réservations dans une transaction ; ça peut être long si l'agence discute avec le client pendant la transaction SQL

Problèmes avec les transactions longues

- Manque de souplesse : si on ne trouve pas de voyage Beijing – Wuhan, on annule tout
- On aurait pu garder le Nice – Paris et essayer de passer par Shanghai pour aller à Wuhan, en annulant seulement le Paris – Beijing

Points de reprise

- On peut désigner des points de reprise dans une transaction : `savepoint nomPoint`
- On peut ensuite annuler toutes les modifications effectuées depuis un point de reprise s'il y a eu des problèmes : `rollback to nomPoint`
- Ça évite d'annuler toute la transaction et permet d'essayer de pallier le problème

Exemple

```
insert into ….;
savepoint p1;
delete from …;
update …;
savepoint p2;
insert into …;
rollback to p2;
commit;
```

2.3/ Accès concurrents sous Oracle

2.3.1/ Introduction

- Oracle, par défaut, ne met aucun verrou pour effectuer une lecture
- Les lectures ne sont pas bloquées par les écritures, et vice-versa
- Pour cela, un historique des modifications est conservé (dans les segments de *rollback*)
- Une transaction est bloquée si elle veut modifier des données qui sont en cours de modification par une autre transaction

- La granularité minimum de blocage est la ligne
- Blocages explicites et implicites
    - Des blocages sont effectués implicitement par certaines commandes (en particulier UPDATE, INSERT et DELETE)
    - Si le comportement par défaut du SGBD ne convient pas pour un traitement, on peut effectuer des **blocages explicites** des lignes ou des tables
    - Les **inter-blocages** sont détectés par le SGBD qui annule un des ordres qui a provoqué l'inter-blocage

2.3.2/ **Blocages explicites**

Granularité des blocages

- 2 types de blocages :
    - au niveau d'une table
    - au niveau d'une ligne ;

Types de blocages Un blocage peut être

exclusif : seul celui qui a bloqué peut modifier ou debloquer les données bloquées

partagé : plusieurs transactions peuvent effectuer en même temps ce type de blocage. C'est un blocage défensif qui marque les données pour interdire qu'une autre transaction ne les bloque dans un mode trop restrictif, ou ne les modifie

Blocages sur les tables

- `LOCK TABLE table IN EXCLUSIVE MODE [NOWAIT]`
  blocage complet de la table (à éviter si possible) ; on peut tout faire dans la table et les autres transactions ne peuvent que lire
- `LOCK TABLE table IN SHARE MODE [NOWAIT]`
  Interdit aux autres transactions toute modification sur la table (sert pour faire des bilans) ; Autorise les autres transactions à bloquer elles aussi en mode SHARE

Blocages partagés sur les lignes

- `SELECT colonnes FROM table WHERE condition FOR UPDATE OF colonnes`

- Ce blocage permet de récupérer des valeurs et de bloquer en même temps les lignes de ces valeurs
- On peut ainsi travailler avec ces valeurs pour préparer leur modification
- Plusieurs transactions peuvent effectuer ce blocage sur des lignes différentes d'une même table

SELECT FOR UPDATE n'interdit pas un blocage en mode partagé de la table mais on est certain que les lignes bloquées ne seront pas modifiées par une autre transaction.

On peut modifier ensuite ces valeurs par UPDATE ou DELETE (après un éventuel déblocage d'un blocage partagé sur la table, effectué par une autre transaction)

2.3.3/ **Blocages implicites**

Les commandes INSERT, UPDATE et DELETE effectuent implicitement un blocage exclusif des lignes modifiées un blocage partagé sur la table qui empêche un blocage de la table par une autre transaction, qui interdirait la modification des données

Annexe complément SQL

Séquence

Créer une séquence

- `CREATE SEQUENCE` *`nom_séquence`* `[INCREMENT BY` *`entier1`*`][START WITH` *`entier2`*`]`
- `create sequence seqdept increment by 10 start with 10`

Utilisation des séquences

- Deux pseudo-colonnes permettent d'utiliser les séquences :
  `CURVAL retourne la valeur courante`
  `NEXTVAL incrémente la séquence et retourne la nouvelle valeur`
- `insert into dept(dept, nomd) values (seqdept.nextval, 'Finances');`

Modification des séquences

```
ALTER SEQUENCE nom_séquence INCREMENT BY entier1
alter sequence seqdept increment by 5
```

Pour Visualiser la valeur d'une séquence

```
Select seqdept.curval from dual
```