

# TD 5 - BDD

15/4/21

## Partie 1

Quel est le rôle qui vous est affecté ?

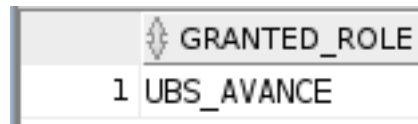
Cette requête nous permet de savoir dans quelle table chercher les rôles/privilèges attribués.

```
select * from dict where comments like '%role%';
```

On peut voir ci-dessous le rôle qui nous est attribué : 'UBS\_AVANCE'

```
select granted_role from DBA_ROLE_PRIVS where grantee = upper('e1800010');
```

Voir pic 1



	GRANTED_ROLE
1	UBS_AVANCE

Figure 1: pic 1

On peut voir ci-dessous le rôle de l'utilisateur 'KESSLER\_BDD': 'UBS\_PROF'

```
select * from DBA_ROLE_PRIVS where grantee = upper('KESSLER_BDD');
```

Voir pic 2



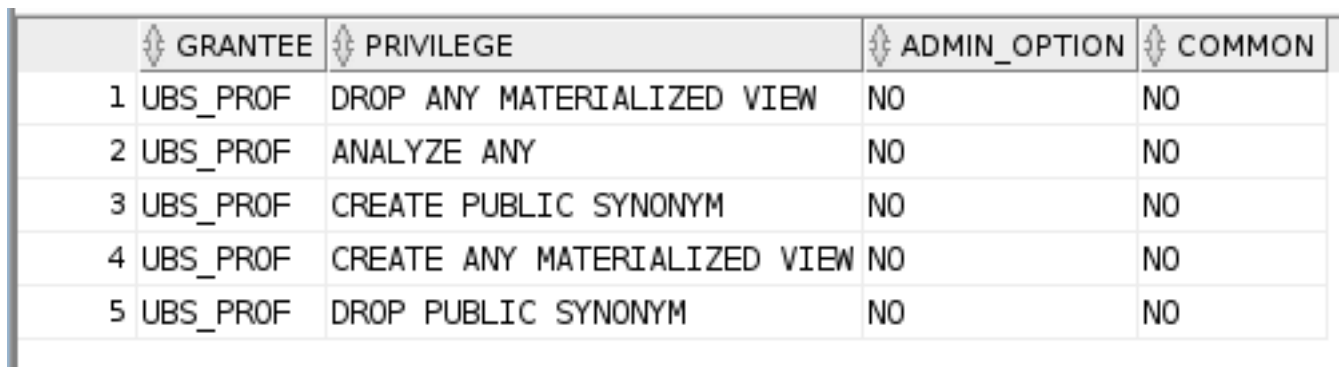
	GRANTEE	GRANTED_ROLE	ADMIN_OPTION	DELEGATE_OPTION	DEFAULT_ROLE	COMMON
1	KESSLER_BDD	UBS_PROF	NO	NO	YES	NO

Figure 2: pic 2

L arbre des privilèges pour cet utilisateur est le suivant, car son rôle est 'UBS\_PROF'.

```
select * from dba_sys_privs where grantee='UBS_PROF';
```

Voir pic 3



	GRANTEE	PRIVILEGE	ADMIN_OPTION	COMMON
1	UBS_PROF	DROP ANY MATERIALIZED VIEW	NO	NO
2	UBS_PROF	ANALYZE ANY	NO	NO
3	UBS_PROF	CREATE PUBLIC SYNONYM	NO	NO
4	UBS_PROF	CREATE ANY MATERIALIZED VIEW	NO	NO
5	UBS_PROF	DROP PUBLIC SYNONYM	NO	NO

Figure 3: pic 3

## Partie 2

### Question 1

Création de la table 'articleTest', puis insertion des valeurs suivantes et validation.

```
DROP TABLE articleTest ;
CREATE TABLE articleTest
(
  idarticle    NUMBER,
  designation   VARCHAR2 (20),
  prixunit     NUMBER (7,2),
  qtestock     NUMBER DEFAULT 0,
  CONSTRAINT pk_articleTest PRIMARY KEY (idarticle)
) ;
INSERT INTO articleTest
VALUES (1, 'classeur', 15.7, 100) ;
INSERT INTO articleTest
VALUES (2, 'cahier', 4.55, 250) ;
INSERT INTO articleTest
VALUES (3, 'stylo', 22.7, 300) ;
INSERT INTO articleTest
VALUES (4, 'chemise', 3.75, 200) ;
INSERT INTO articleTest
VALUES (5, 'crayon', 7.8, 1000) ;
COMMIT;
```

Effectuez une série de mises à jour sur votre table articleTest (INSERT, DELETE, UPDATE) :

```
INSERT INTO articleTest VALUES (6, 'ciseaux', 10.5, 200) ;
DELETE FROM articleTest WHERE idArticle=1 ;
UPDATE articleTest SET prixunit=prixunit*2 WHERE idArticle=2 ;
```

	IDARTICLE	DESIGNATION	PRIXUNIT	QTESTOCK
1	2	cahier	9,1	250
2	3	stylo	22,7	300
3	4	chemise	3,75	200
4	5	crayon	7,8	1000
5	6	ciseaux	10,5	200

Voir pic 4

On visualise ce qu'on vient d'insérer de modifier, puis on annule cette mise à jour avec un 'ROLLBACK' et on peut revisualiser cette annulation ensuite.

```
select * from articleTest;
ROLLBACK;
select * from articleTest;
```

Voir pic 5

	IDARTICLE	DESIGNATION	PRIXUNIT	QTESTOCK
1	1	classeur	15,7	100
2	2	cahier	4,55	250
3	3	stylo	22,7	300
4	4	chemise	3,75	200
5	5	crayon	7,8	1000

Figure 4: pic 5

## Question 2

Je donne les droits suivant à l'utilisateur e1803993:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON articleTest TO E1803993;
```

Output:

```
-- succès de l'élément Grant.
```

Insertion de tuple dans la table E1803993 :

```
select * from E1803993.articleTest;
INSERT INTO E1803993.articleTest VALUES (6, 'ciseau', 10.5, 200);
INSERT INTO E1803993.articleTest VALUES (7, 'couteau', 10.5, 200);
COMMIT;
-- sans faire de commit, l'utilisateur ne verra pas l'ajout que j'ai fais
```

Je regarde ce que l'autre utilisateur a rajouté dans ma table :

```
select * from articleTest;
```

Le tuple 'fourchette' a été ajouté après que l'utilisateur ait fait une validation (COMMIT).

J'insère un tuple dans la table articleTest de E1803993. E1803993 se positionne en mode READ ONLY.

```
INSERT INTO E1803993.articleTest VALUES (8, 'cuillère', 10.5, 200);
-- 1 ligne insérée
select * from E1803993.articleTest;
-- la ligne est bien insérée de mon côté
-- e1803993 se met en read only avec la commande suivante (que je n'exécute pas) :
-- ALTER TABLE E1803993.articleTest READ ONLY;
-- et elle peut voir l'insertion que j'ai faite sans que j'ai eu besoin de faire de COMMIT
```

Le fait de s'être mis en mode READ ONLY lui permet de voir les modifications que j'ai faites, et si je fais un COMMIT, cela annule son mode READ ONLY.

Après validation de la transaction :

```
COMMIT;
select * from E1803993.articleTest;
-- je ne vois aucun changement, le tuple inséré juste avant est présent
-- rien ne change du côté de e1803993, car le commit a validé l'insertion faite avant.
```

Note: il se peut que Oracle bug, cela arrive assez souvent d'ailleurs.

### Pose de verrou implicite

Je modifie un tuple de la

```
UPDATE E1803993.articleTest SET prixunit=0 WHERE idArticle=8;
COMMIT; -- on sort du mode read only
-- impossible d'update la table de e1803993, nous n'avons pas les droits
-- oracle nous dit que seul Oracle peut modifier les tables qui sont en read only, pourtant nous ne sommes pas Oracle
```

Voir pic 6

C'est une erreur normale, et si e1803993 essaie de faire la même chose, elle a la même erreur, c'est normal.

```
UPDATE E1803993.articleTest SET prixunit=0 WHERE idArticle=5;
```

Même erreur, c'est normal.

### Pose de verrou explicite

Je verrouille ma table :

```
LOCK TABLE articleTest IN EXCLUSIVE MODE;
-- Succès de l'élément Lock.
```

Je modifie ma table :

```
Erreur commençant à la ligne: 128 de la commande -
UPDATE E1803993.articleTest SET prixunit=0 WHERE idArticle=8
Erreur à la ligne de commande: 128 Colonne: 17
Rapport d'erreur -
Erreur SQL : ORA-12081: opération de mise à jour interdite sur la table "E1803993"."ARTICLETEST"
12081. 00000 - "update operation not allowed on table \"%s\".\"%s\""
*Cause:      An attempt was made to update a read-only materialized view.
*Action:     No action required. Only Oracle is allowed to update a
              read-only materialized view.
Validation (commit) terminée.
```

Figure 5: pic 6

```
UPDATE articleTest SET prixunit=0 WHERE idArticle=2;
COMMIT;
select * from articleTest;
-- ma modification est bien présente
```

U2 fait une modification sur ma table, je peux la voir :

```
select * from articleTest;
-- tuple 9 rajouté par e1803993
```

Si E1803993 essaie de Lock ma table, ça marche, même résultat que quand je l'ai fait.

Note : Normalement e1803993 ne devrait pas pouvoir faire d'insertion parce que ma table est lock, mais c'est un problème venant d'Oracle, nous n'y pouvons rien. Il aurait peut-être fallu se déconnecter et se reconnecter, mais nous n'avons pas le temps de le faire, ni la certitude que cette méthode que vous nous avez dites pourrait marcher.

On peut se verrouiller mutuellement les tables, mais nous y avons encore le droit d'accès (que ce soit en lecture ou en écriture).

## Interblocages

Comment pourrait-on résoudre une situation d'interblocage ?

Nous pouvons déjà définir ce qu'est une situation d'interblocage. Ce genre de situation intervient lorsque deux processus tentent de modifier les mêmes données au même moment. Le SGBD, ici Oracle, bloque ces processus. Pour résoudre le problème, le sgbd utilise l'algorithme FIFO (First In First Out), qui signifie comme son nom l'indique que si 2 instructions sont lancées en même temps, c'est la 1ère lancée qui sera la première à être réalisée. Il y aura donc un temps d'attente de l'autre instruction. Et comme pour tout autre type de requête, cela peut être annulé via un ROLLBACK. Et on peut aussi tuer (kill) le processus qui cause le blocage du sgbd.