

Gestion des Transactions



Pr. A. Massaq

Introduction transactions

En BD 2 problèmes :

- De multiples utilisateurs doivent pouvoir accéder à la base de donnée en même temps → problème **d'accès concurrents**
- De nombreuses et diverses **pannes** peuvent apparaître. Il ne faut pourtant pas perdre les données...

La Gestion de transactions répond à ces problèmes

Notion de transaction

- Exemple : achat baguette
 - donne baguette
 - payer
 - rend monnaie

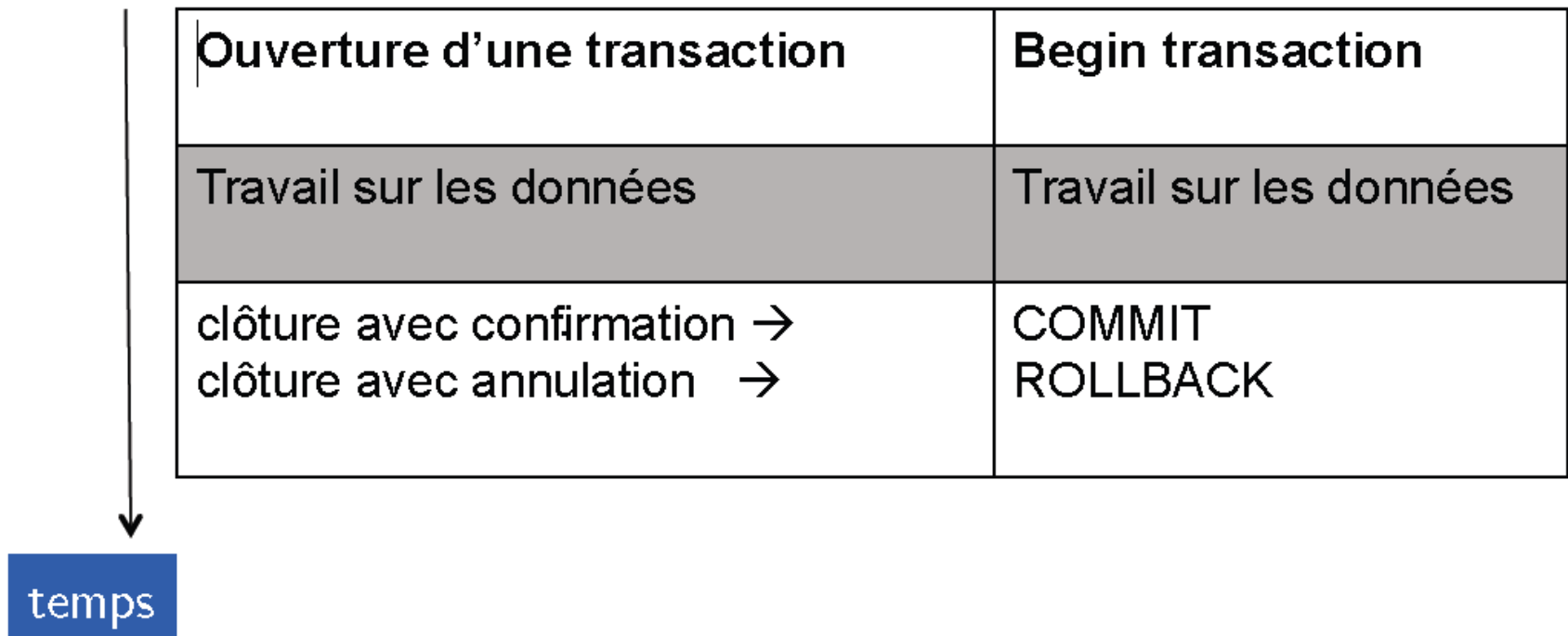
Notion de transaction

Une transaction est une suite d'opérations (LMD) interrogeant et/ou modifiant la BD, pour laquelle l'ensemble des opérations doit être soit **validé**, soit **annulé**.

- Toute la transaction est réalisée ou rien ne l'est
 - **Validation** : toute la transaction est prise en compte
 - **Avortement ou annulation** : la transaction n'a aucun effet
- Sous oracle
 - Début d'une transaction : ordre SQL
 - Fin d'une transaction : validation(**Commit**) ou annulation (**Rollback**)

Notion de transaction

Une transaction est constituée de trois primitives:
opérations pour délimiter



Ouverture d'une transaction	Begin transaction
Travail sur les données	Travail sur les données
clôture avec confirmation → clôture avec annulation →	COMMIT ROLLBACK

Pseudo-code d'une transaction de transfert bancaire

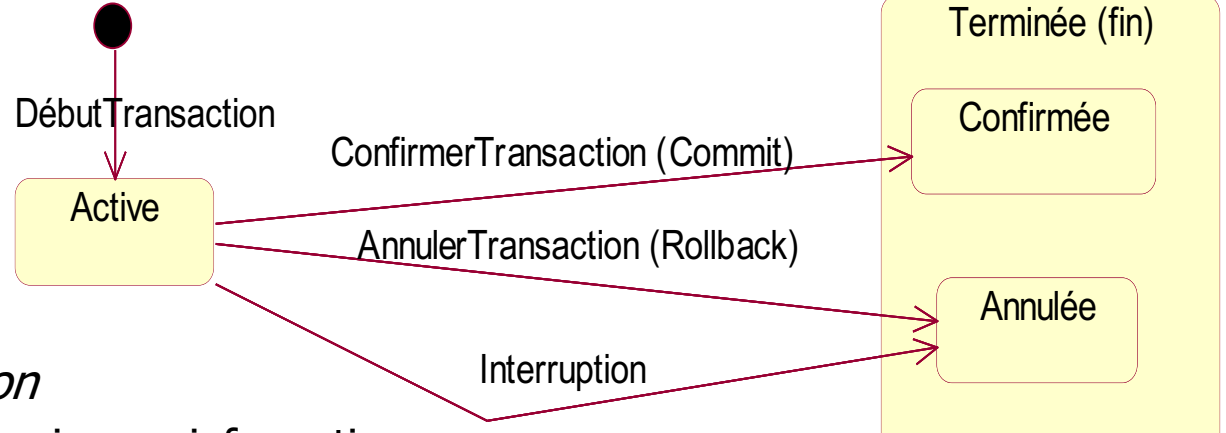
```
DébutTransaction ;  
Lire (SoldeCompte1, s1) ;  
s1 := s1 - 100 ;  
Écrire (s1, SoldeCompte1) ;  
SI s1 < 0  
    AnnulerTransaction ;  
SINON  
    Lire (SoldeCompte2, s2) ;  
    s2 := s2 + 100 ;  
    Écrire (s2, SoldeCompte2) ;  
    ConfirmerTransaction ;  
FINSI ;
```

Traduction en sql

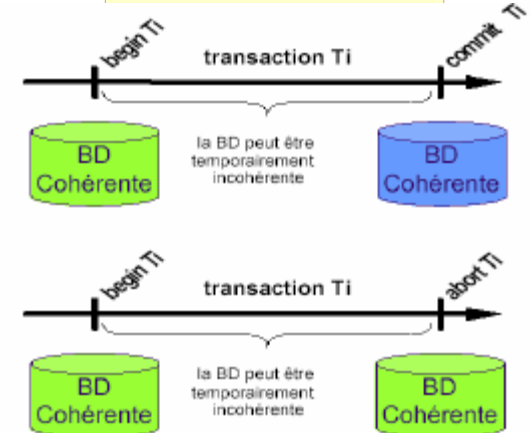
UPDATE Compte1 Val = Val -100

UPDATE Compte2 Val = Val + 100

État d'une transaction



- *DébutTransaction*
 - identificateur unique + informations
 - état *actif*
 - *consommation de ressource*
- *ConfirmerTransaction*
 - état *confirmé*
 - point de confirmation
 - enregistrement persistant (journal)
 - *libération de ressources*
 - *confirmer le plus tôt possible pour éviter de monopoliser les ressources*
- *AnnulerTransaction* ou interruption suite à une erreur (faute)
 - état *annulé*
 - effets doivent être « défaits »
- *Fin = confirmé ou annulé*



Exemples de fautes

- Erreur locale à une transaction
- Annulation par le SGBD (par exemple suite à la détection d'un interblocage)
- Erreur du SGBD
- Erreur du système d'exploitation
- Panne de matériel
- etc.

Accès concurrent

- Il y a un accès concurrent lorsque plusieurs utilisateurs (transactions) accèdent en même temps à la même donnée dans une base de données.
- La concurrence d'accès aux données induit des problèmes inévitables.

Le problème est que ces transactions sont susceptibles de s'exécuter en même temps.

Problèmes posés par les accès concurrents

– Perte de mise à jour :

T1 et T2 modifient simultanément A

T_1	T_2	BD
		$A = 10$
read A		
	read A	
$A = A + 10$		
write A		$A = 20$
	$A = A + 50$	
	write A	$A = 60$

Les modifications effectuées par T1 sont perdues

-Lecture impropre

T_1	T_2	BD
		$A + B = 200$
		$A = 120$ $B = 80$
read A		
$A = A - 50$		
write A		$A = 70$
	read A	
	read B	
	display A + B (150 est affiché)	
read B		
$B = B + 50$		
write B		$B = 130$

T1 doit validé opération A-50

T2 lit une valeur de A non validée, affiche une valeur incohérence (T1 peut annuler l'opération A-50) → **Accès à des données non validées**

Lecture incohérente

T2 lit deux valeurs de A différentes

T_1	T_2	BD
		$A = 10$
	read A (10 est lu)	
$A = 20$		
write A		$A = 20$
	read A (20 est lu)	

Si au cours d'une même transaction T2 accède deux fois à la valeur d'un tuple alors que ce dernier est, entre les deux, modifié par une autre transaction T1, alors la lecture de A est inconsistente. Ceci peut entraîner des incohérences

Le verrouillage

- Une solution générale à la gestion de la concurrence est une technique très simple appelée verrouillage.

- **Verrou**

Poser un verrou sur un objet par une transaction signifie rendre cet objet inaccessible aux autres transactions.

Synonymes : Lock

- **Déverrouillage** Lorsqu'une transaction se termine (COMMIT ou ROLLBACK) elle libère tous les verrous qu'elle a posé.

Synonymes : *Unlock*

Verrouillage provoqués par l'utilisateur : Commande LOCK TABLE

Le choix d'Oracle concernant la pose de verrous peut être remis en cause par l'utilisateur à l'aide de la commande LOCK TABLE, en voici la syntaxe :

```
LOCK TABLE < liste des noms de tables > IN < type de verrou > MODE ;
```

Les types de verrous sont les verrous présentés plus haut : EXCLUSIVE, SHARE, ROW SHARE

Exemple : LOCK TABLE médicament IN EXCLUSIVE MODE ;

Si un autre utilisateur contourne un verrouillage, en établissant son propre verrou, la commande normalement attend jusqu'à ce que l'autre utilisateur fasse un COMMIT ou un ROLLBACK. Il est cependant possible d'utiliser l'option NOWAIT pour que le verrou ne soit pas mis en attente.

Exemple: LOCK TABLE médicament IN SHARE UPDATE MODE NOWAIT;

Transactions en sql

Début d'une transaction

BEGIN TRANSACTION (ou **BEGIN**) ;

Cette syntaxe est optionnelle (voire inconnue de certains SGBD), une transaction étant débutée de façon implicite dès qu'instruction est initiée sur la BD.

En oracle:

- La commande BEGIN; ou BEGIN TRANSACTION; ne peut pas être utilisé sous Oracle (la commande BEGIN est réservée à l'ouverture d'un bloc PL/SQL).
- Toute commande SQL LMD (INSERT, UPDATE ou DELETE) démarre par défaut une transaction, la commande BEGIN TRANSACTION est donc implicite.

Transactions en sql

COMMIT et ROLLBACK

Fin correcte d'une transaction

COMMIT TRANSACTION (ou **COMMIT**) ;

Cette instruction SQL signale la fin d'une transaction couronnée de succès

- **COMMIT** permet d'enregistrer en base toutes les modifications (LMD) effectuées au cours de la transaction
- **ROLLBACK** : permet d'annuler en base toutes les modifications (LMD) effectuées au cours de la transaction.

Exemple : Transaction sous Oracle en **PL/SQL**

```
BEGIN
```

```
INSERT INTO test (a) VALUES (1);
```

```
COMMIT;
```

```
END;
```

SAVEPOINT

- **Cette instruction permet de placer une étiquette savepoint dans le corps du code.**
- **Marque le point de validation**
- **Elle permet au traitement d'annuler, avec l'instruction ROLLBACK, les modifications effectuées à partir de cette étiquette**

SAVEPOINT

- **Exemple :**

Application

```
BEGIN
```

```
    INSERT INTO dept values( 43, 'info', 'Paris' );
```

```
    SAVEPOINT mise_a_jour;
```

```
    INSERT INTO dept values( 44, 'gestion', 'Lille' );
```

```
    UPDATE EMP Set sal = sal + 1000 Where empno > 7900;
```

```
    ROLLBACK TO SAVEPOINT mise_a_jour;
```

```
    Commit;
```

```
End ;
```

```
/
```

Annulation porte sur toutes les modifications effectuées à partir de l'étiquette nom savepoint

- MySQL travaille par défaut en mode ***autocommit***, c'est-à-dire que chaque instruction SQL du LMD qui se termine avec succès est automatiquement validée.
- Pour annuler l'autovalidation en mysql :
SET autocommit=0
(1 par défaut, 0 à adopter pour contrôler une transaction)

- Oracle

- Sous Oracle SQL*Plus : SET AUTOCOMMIT ON et SET AUTOCOMMIT OFF

- Sous Oracle SQL Developer : Menu Outils > Préférences > Base de données > Paramètres de feuille de calcul > Validation automatique dans une feuille de calcul

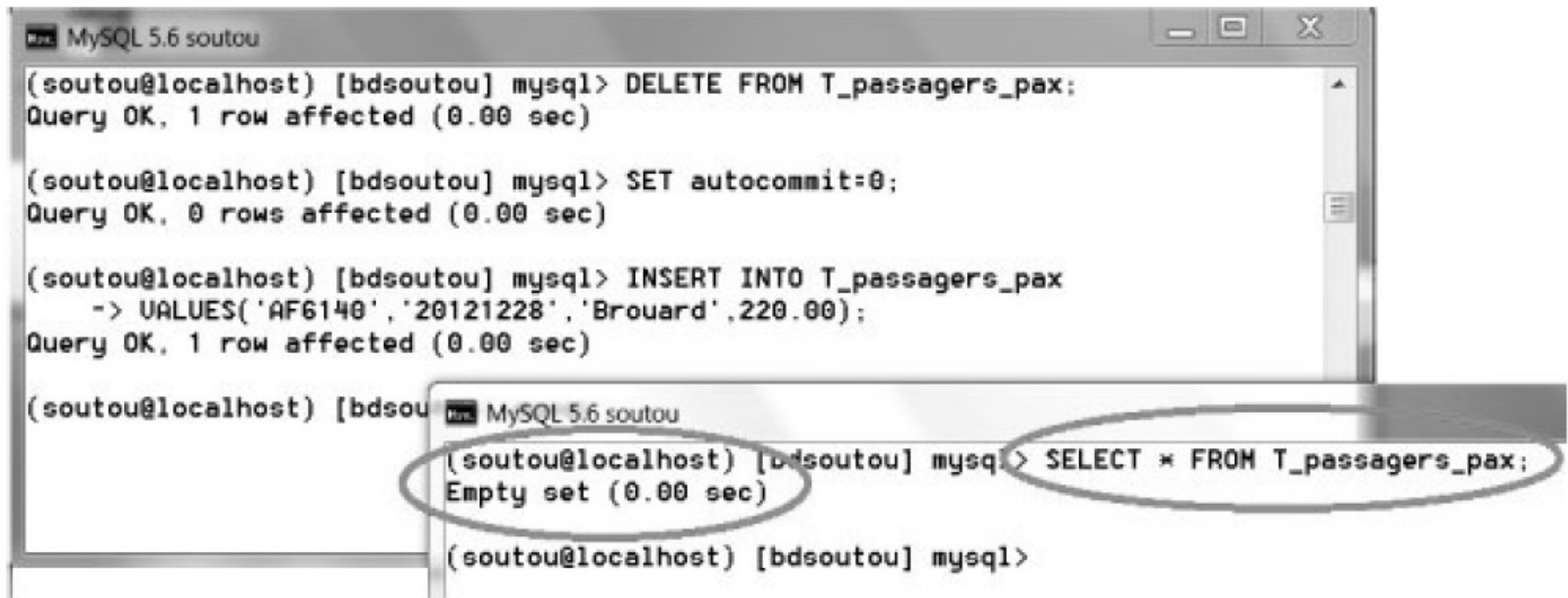
- PostgreSQL

- \set AUTOCOMMIT on

- \set AUTOCOMMIT off

Transaction non validée

~



The screenshot shows a MySQL 5.6 terminal window with the following commands and output:

```
MySQL 5.6 soutou
(soutou@localhost) [bdsoutou] mysql> DELETE FROM T_passagers_pax;
Query OK, 1 row affected (0.00 sec)

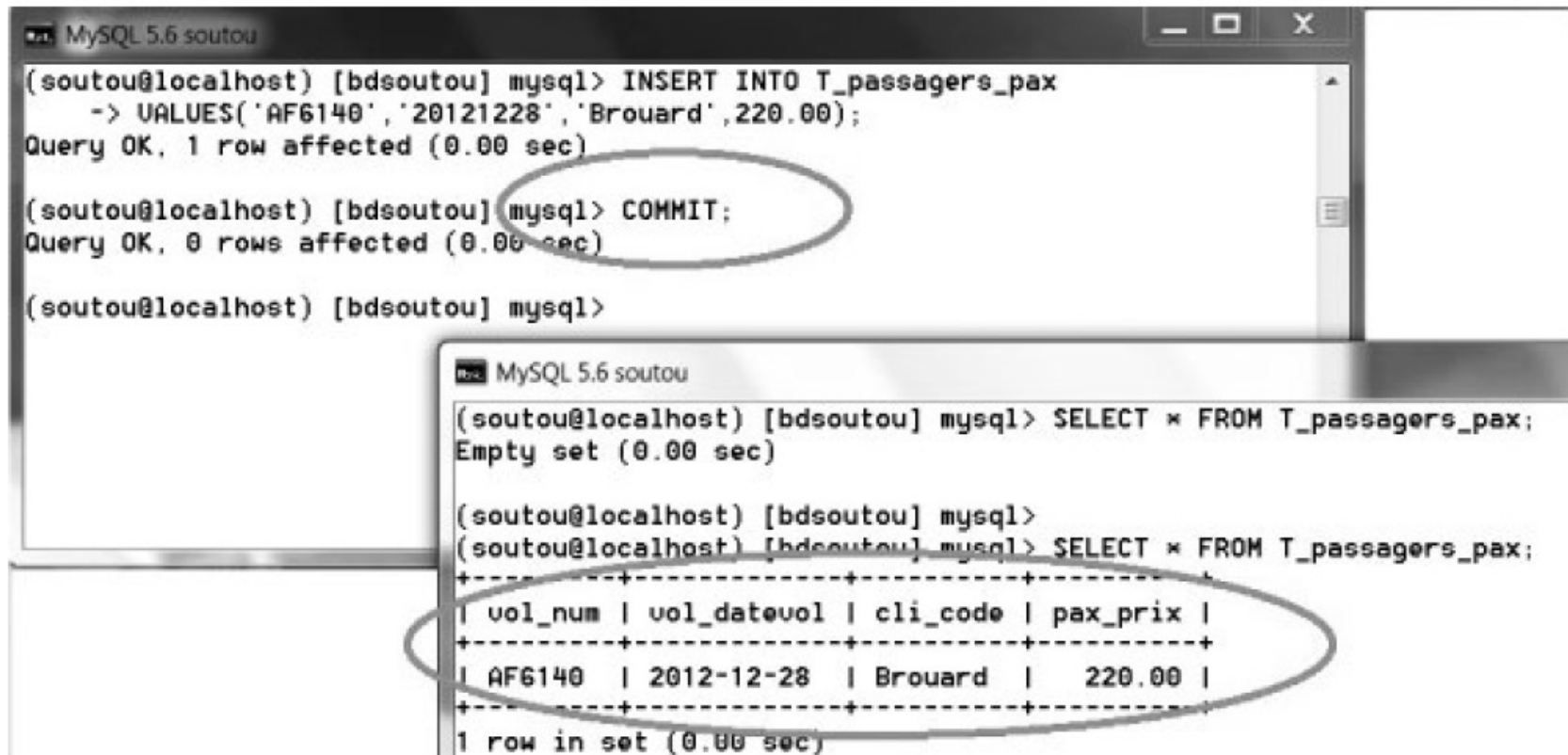
(soutou@localhost) [bdsoutou] mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)

(soutou@localhost) [bdsoutou] mysql> INSERT INTO T_passagers_pax
-> VALUES('AF6140','20121228','Brouard',220.00);
Query OK, 1 row affected (0.00 sec)

(soutou@localhost) [bdsoutou] mysql> SELECT * FROM T_passagers_pax;
Empty set (0.00 sec)
```

The last command and its output, "Empty set (0.00 sec)", are circled in the image, indicating that the transaction was not committed.

Transaction validée



```
MySQL 5.6 soutou
(soutou@localhost) [bdsoutou] mysql> INSERT INTO T_passagers_pax
-> VALUES('AF6140', '20121228', 'Brouard', 220.00);
Query OK, 1 row affected (0.00 sec)

(soutou@localhost) [bdsoutou] mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

(soutou@localhost) [bdsoutou] mysql>

MySQL 5.6 soutou
(soutou@localhost) [bdsoutou] mysql> SELECT * FROM T_passagers_pax;
Empty set (0.00 sec)

(soutou@localhost) [bdsoutou] mysql>
(soutou@localhost) [bdsoutou] mysql> SELECT * FROM T_passagers_pax;
+-----+-----+-----+-----+
| vol_num | vol_datevol | cli_code | pax_prix |
+-----+-----+-----+-----+
| AF6140  | 2012-12-28  | Brouard  | 220.00   |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```


-Il n'est pas possible d'invalider par ROLLBACK une commande LDD (CREATE, ALTER, DROP...).

Fin d'une transaction

Une transaction peut être terminée de plusieurs façons :

- L'utilisateur exécute un COMMIT ou un ROLLBACK sans point de sauvegarde (nous verrons les points de sauvegarde plus loin).
- Certaines commandes provoquent des COMMIT automatiquement, en voici la liste :

```
ALTER  
AUDIT  
NOAUDIT  
COMMENT  
CONNECT  
CREATE  
DISCONNECT  
DROP  
EXIT  
GRANT  
REVOKE  
QUIT  
RENAME
```