

*Solutions proposées TP*

*BDA : TP5*

*Enseigné par :*

**Samir YUCEF**

*Réalisé par l'étudiant :*

- Lucas ZHENG

[lucas.zheng@edu.univ-paris13.fr](mailto:lucas.zheng@edu.univ-paris13.fr)

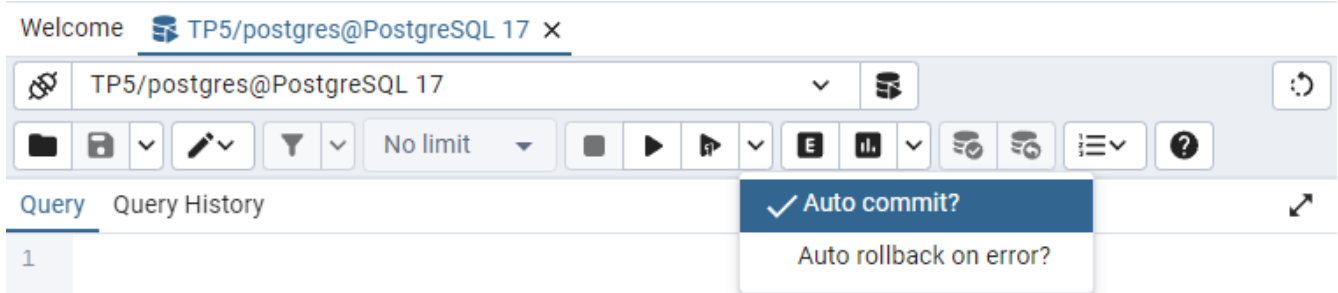
# Table des questions

Contexte.....	2
Exercice 1.....	2
Exercice 2.....	6
1. Isolation des transactions.....	6
2. Isolation incomplète = incohérence possible.....	8
3. Isolation complète = blocage et rejet des transactions possibles.....	10

# Contexte

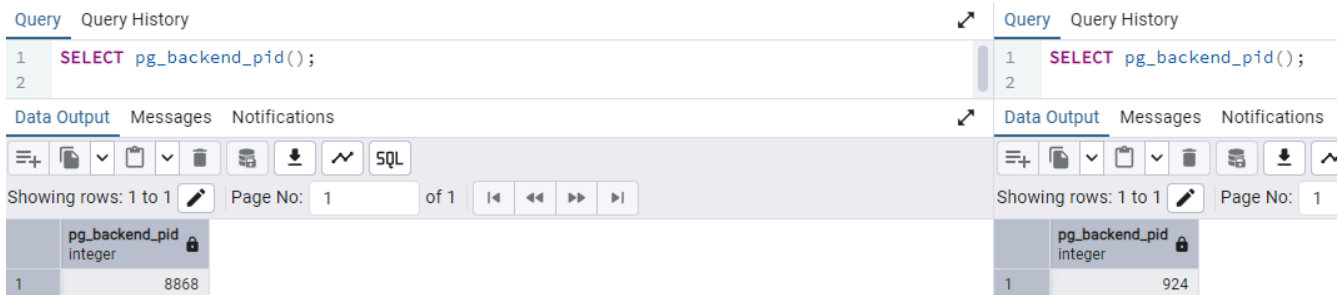
Le travail réalisé dans le cadre de ce TP a été effectué avec PostgreSQL.  
Il existe certaines différences de syntaxe entre PostgreSQL et Oracle.

En particulier, l'autocommit on ou off est défini directement sur l'option du bouton d'exécution du script.



## Exercice 1

Commencer par ouvrir deux sessions (S1 et S2) et exécuter la commande suivantes dans chacune des sessions (dans la suite de cet exercice, et à chaque connexion (ouverture de session), commencer par exécuter cette commande) :



## Etape 1 / 2 : Avant le ROLLBACK

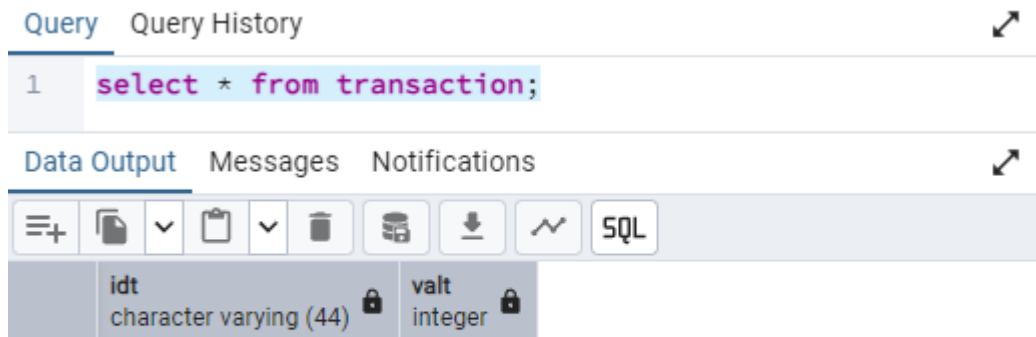
## Etape 2 / 2 : Après le ROLLBACK

3

Insérer à nouveau dans transaction quelques lignes et clore avec la commande quit;. Consulter le contenu de la table, en utilisant la session S1. Que s'est-il passé ?

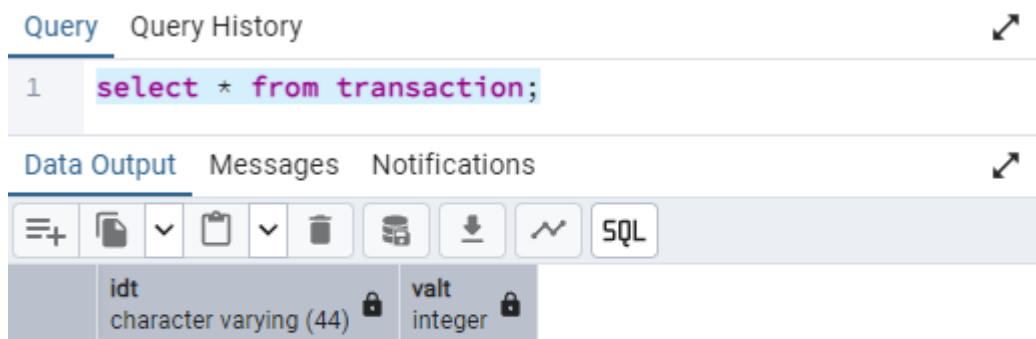
Lorsque j'insère des lignes dans la session S2 puis que je quitte (quit;) sans avoir émis de COMMIT, PostgreSQL, en mode d'isolation READ COMMITTED, ne rend ces lignes jamais visibles aux autres sessions et, de plus, procède à un rollback implicite à la fermeture de S2.

Quand je me reconnecte en S2 (une session indépendante avec un identifiant différent de l'ancienne session 2 ), je ne les retrouve donc plus.

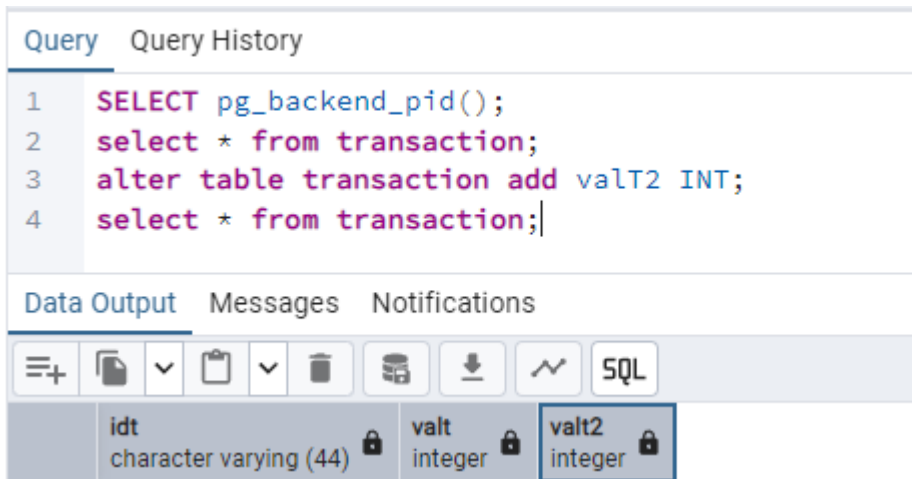


Dans la session S1, insérer quelques lignes et fermer brutalement sqlplus. Reconnecter à nouveau sur votre compte. Les données saisies ont-elles été préservées ?

Non car cela rejoint l'explication précédente sur les identifiants distincts entre l'ancienne et la nouvelle session.



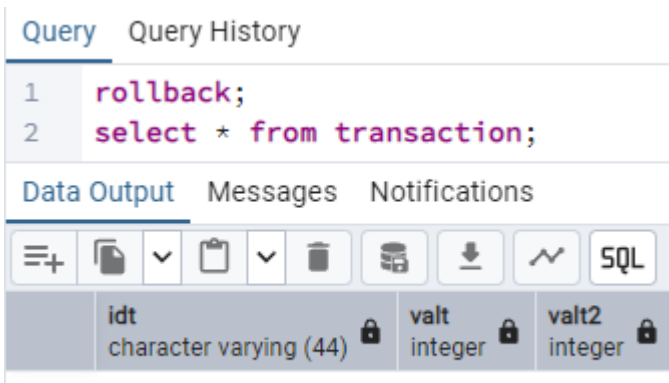
Dans une nouvelle session, insérer quelques lignes et modifier la structure de la table transaction, en y ajoutant par exemple l'attribut val2transaction de type NUMBER(10). Exécuter la commande ROLLBACK. Que s'est-il passé ?



```
1 SELECT pg_backend_pid();
2 select * from transaction;
3 alter table transaction add valT2 INT;
4 select * from transaction;
```

Column	Type
idt	character varying (44)
valt	integer
valt2	integer

L'attribut valt2 est bien ajouté, mais il persiste après un ROLLBACK sur la capture suivante. L'hypothèse serait que l'opération est effectuée sur la structure de la table et non sur ses données, ce qui entraîne une validation automatique (auto-commit) de l'opération.



```
1 rollback;
2 select * from transaction;
```

Column	Type
idt	character varying (44)
valt	integer
valt2	integer

Conclure : définir de manière précise qu'est-ce qu'une session, une transaction et comment valider une transaction ou l'annuler.

- SESSION : la période de connexion entre un client (psql, pgAdmin, application, etc.) et le serveur PostgreSQL, depuis l'authentification (login) jusqu'à la déconnexion (logout).
- TRANSACTION : Série d'opérations SQL avant de finir par un COMMIT ou un ROLLBACK
- VALIDER : COMMIT rend permanentes et visibles à toutes les autres sessions les modifications effectuées depuis le début de la transaction. ( sauf si le niveau d'isolation est Read uncommitted auquel cas on peut faire la lecture sale )
- ANNULER : ROLLBACK annule les opérations SQL de la transaction en cours, à l'exception des modifications effectuées sur la structure des tables, qui sont automatiquement validées (auto-commit).

## Exercice 2

### 1. Isolation des transactions

Effectuer une réservation pour un client et ne validez pas encore cette transaction (T1)

On constate bien d'après les captures ci-dessous que la donnée insérée par la session 1 n'est visible que par cette dernière.

Welcome TP5/postgres@PostgreSQL 17\* x TP5/postgres@Po... x

TP5/postgres@PostgreSQL 17

Query Query History

```
1 insert into client values ('1','session1',1);
2 select * from client;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	idclient character varying (44)	prenomclient character varying (11)	nbrplacesreserveesclient integer
1	1	session1	1

Welcome TP5/postgres@Po... x TP5/postgres@PostgreSQL 17\* x

TP5/postgres@PostgreSQL 17

Query Query History

```
1 select * from client;
```

Data Output Messages Notifications

	idclient character varying (44)	prenomclient character varying (11)	nbrplacesreserveesclient integer
--	------------------------------------	--	-------------------------------------

Effectuez un ROLLBACK de la transaction en cours (T1).

On constate bien d'après les captures ci-dessous que tout est revenu à l'état initial et que le COMMIT ne change rien sans opération après le ROLLBACK.

Welcome TP5/postgres@PostgreSQL 17\* x TP5/postgres@Po... x

TP5/postgres@PostgreSQL 17

Query Query History

```
1 rollback;
2 select * from client;
```

Data Output Messages Notifications

idclient	prenomclient	nbrplacesreserveesclient
character varying (44)	character varying (11)	integer

Welcome TP5/postgres@PostgreSQL 17\* x TP5/postgres@Po... x

TP5/postgres@PostgreSQL 17

Query Query History

```
1 commit; select * from client;
```

Data Output Messages Notifications

idclient	prenomclient	nbrplacesreserveesclient
character varying (44)	character varying (11)	integer



Recommencez la transaction T1 et validez, cette fois, les mises à jour (COMMIT).

On constate bien que le ROLLBACK après le COMMIT ne change rien car le COMMIT a été effectué donc l'insertion de la donnée est définitive.

Ainsi, la session 2 voit bien cette donnée insérée.

Welcome TP5/postgres@PostgreSQL 17\* x TP5/postgres@Po... x

TP5/postgres@PostgreSQL 17

Query Query History

```
1 insert into client values ('1','session1',1);
2 commit; rollback;
3 select * from client;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	idclient character varying (44)	prenomclient character varying (11)	nbrplacesreserveescleint integer
1	1	session1	1

Welcome TP5/postgres@Po... x TP5/postgres@PostgreSQL 17\* x

TP5/postgres@PostgreSQL 17

Query Query History

```
1 select * from client;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	idclient character varying (44)	prenomclient character varying (11)	nbrplacesreserveescleint integer
1	1	session1	1

## 2. Isolation incomplète = incohérence possible

Les étapes sont différentes mais préservent l'ordre des opérations pour des raisons de visibilité avec les captures d'écran.

on effectue les deux sélections pour T1 et T2 (vol) ;

The image shows two side-by-side screenshots of a PostgreSQL client interface. Both windows are titled 'TP5/postgres@PostgreSQL 17\*'. The left window shows a query 'select \* from vol;' and its result set. The right window shows the same query and result set. The result set has three columns: 'idvol' (character varying (44)), 'capacitevol' (integer), and 'nbrplacesreservesvol' (integer). The data row shows 'vol1', '10', and '0'.

	idvol character varying (44)	capacitevol integer	nbrplacesreservesvol integer
1	vol1	10	0

on effectue les deux sélections et deux mises à jour pour T1 et T2 (client) ;

The image shows two side-by-side screenshots of a PostgreSQL client interface. Both windows are titled 'TP5/postgres@PostgreSQL 17\*'. The left window shows a query 'insert into client values ('client1','session1',2);' followed by 'select \* from client;'. The right window shows a query 'insert into client values ('client2','session2',3);' followed by 'select \* from client;'. The result set has three columns: 'idclient' (character varying (44)), 'prenomclient' (character varying (11)), and 'nbrplacesreservescleint' (integer). The data row shows 'client1', 'session1', and '2'.

	idclient character varying (44)	prenomclient character varying (11)	nbrplacesreservescleint integer
1	client1	session1	2

on effectue une mise à jour de T1 ( vol ), et on valide puis on effectue une mise à jour de T2 ( vol ), et on valide ;

The image displays four screenshots of a PostgreSQL client interface, arranged in a 2x2 grid. Each screenshot shows a query being executed and its results in a table format.

**Top Left Screenshot:** The query is `update vol set nbrPlacesReserveesVol = (0+2) where idVol = 'vol1'; commit; select * from vol;`. The results table has columns `idvol`, `capacitevol`, and `nbrplacesreserveesvol`. The first row shows `vol1`, `10`, and `2`.

**Top Right Screenshot:** The query is `update vol set nbrPlacesReserveesVol = (0+3) where idVol = 'vol1'; commit; select * from vol;`. The results table shows `vol1`, `10`, and `3`.

**Bottom Left Screenshot:** The query is `select * from client;`. The results table has columns `idclient`, `prenomclient`, and `nbrplacesreserveesclient`. The first two rows show `client1` with `session1` and `client2` with `session2`, both with a value of `2` in the `nbrplacesreserveesclient` column.

**Bottom Right Screenshot:** The query is `select * from client;`. The results table shows the same two rows as the bottom left, but the `nbrplacesreserveesclient` column now shows `2` for `client1` and `3` for `client2`.

On constate bien que les deux clients ont bien réservé  $2+3 = 5$  billets, mais que seulement 3 billets ont été réservés pour le vol.

### 3. Isolation complète = blocage et rejet des transactions possibles

On peut faire la lecture de données dans les deux sessions même si elles ont posé chacune un verrou mais ce verrou est partagé.

Après l'écriture de la session 2 dans la ligne d'

The left screenshot shows a query window with the following SQL code:

```
1 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
2 select * from client where idclient = 'client1';
```

The right screenshot shows a query window with the following SQL code:

```
1 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
2 select * from client where idclient = 'client1';
3 update client set prenomclient = 'newsession2' where idclient = 'client1';
4
```

Both screenshots show the 'Messages' tab with the following output:

SET

Query returned successfully in 44 msec.

UPDATE 1

Query returned successfully in 41 msec.

D'après la capture d'écran ci-dessous, la session 1 est mise en attente car la session 2 a posé un verrou sur la donnée d' en faisant une écriture sur cette donnée.

The left screenshot shows a query window with the following SQL code:

```
1 update client set prenomclient = 'newsession1' where idclient = 'client1';
2 commit;
```

The right screenshot shows a query window with the following SQL code:

```
1 select * from client;
```

Both screenshots show the 'Data Output' tab with the following output:

Showing rows: 1 to 2 of 1

idclient	client	nbplacesreservesclient
1	client1	session1
2	client2	session2

Total rows: 2

Waiting for the query to complete... 00:02:20.527

CRLF Ln 2, Col 8

The right screenshot shows the 'Data Output' tab with the following output:

Showing rows: 1 to 2 of 1

idclient	client	nbplacesreservesclient
1	client2	session2
2	client1	newsession2

Total rows: 2

Query complete 00:00:00.046

CRLF Ln 1, Col 22

Ainsi, vu que la session 2 a posé le verrou exclusif sur la donnée d', la session 1 restera en attente indéfiniment jusqu'à que la session 2 fait un commit ou un rollback.

Le rejet des transactions aurait été possible si on avait défini un dépassement de délai des requêtes mais ce n'est pas le cas dans cet essai.

Donc, le niveau d'isolation serializable ne suit pas exactement le verrouillage à deux phases vu en cours car la session 2 ne devrait pas écrire sur la donnée d' car cette dernière a déjà un verrou partagé.