Alexis N3 Informatique

PLESSIER

27/12/2020

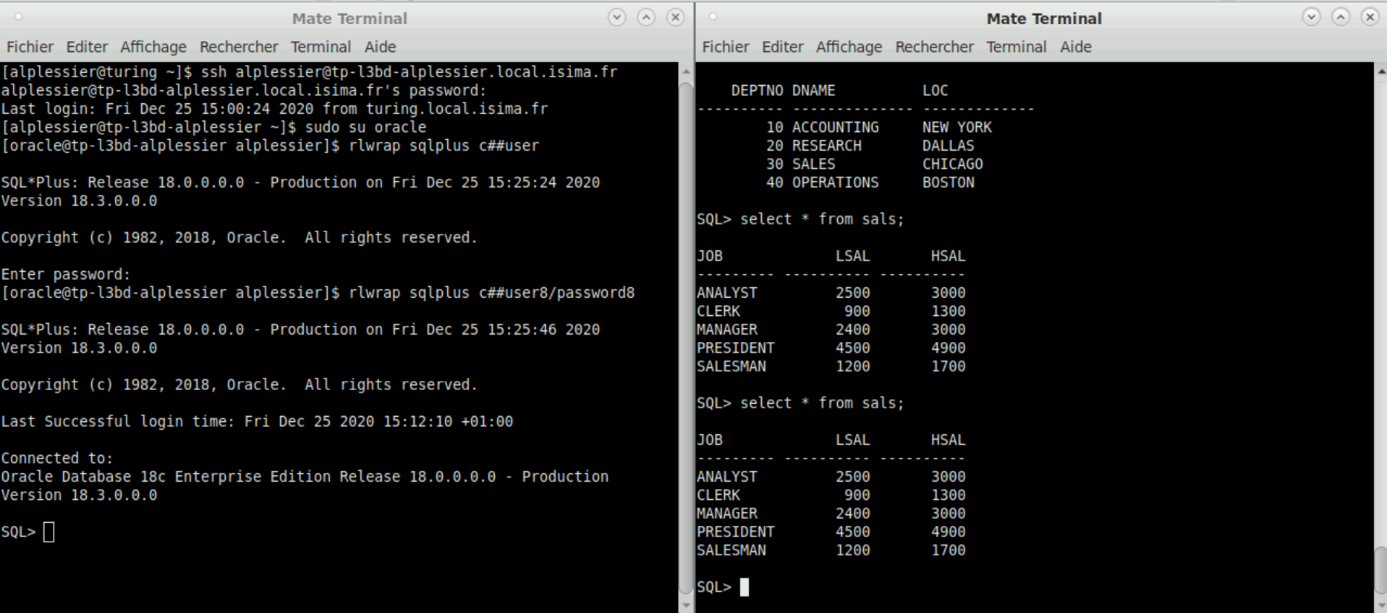
Implémentation des SGBD

Rendu TP1

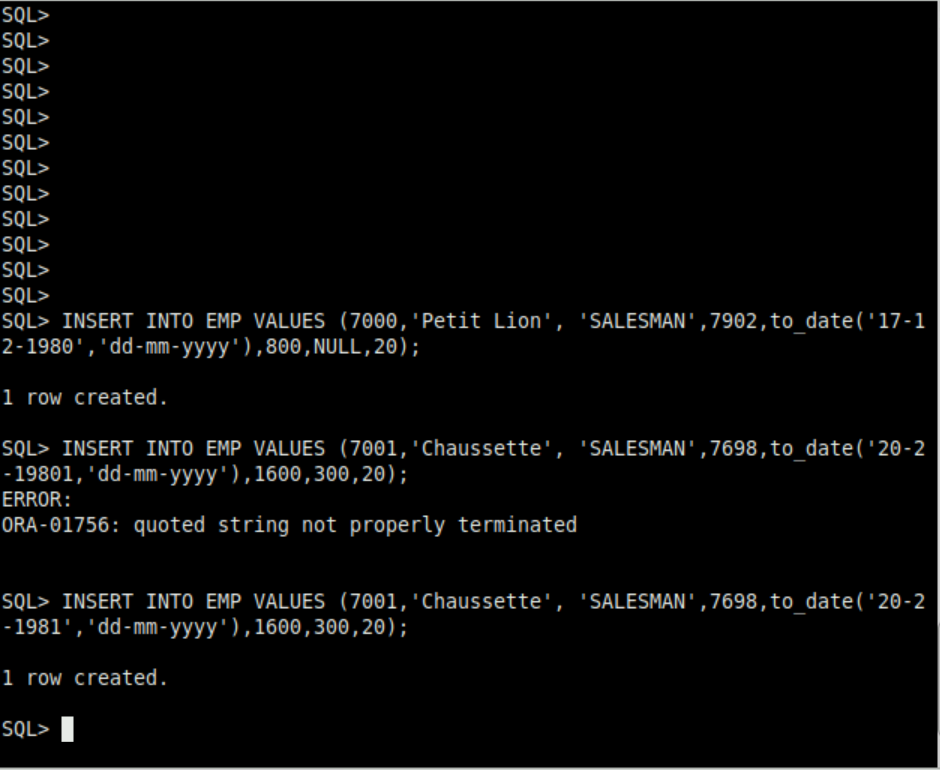
Exercice 1 :

1) Ouverture des deux sessions avec mon login « alplessier » et mon mot de passe.

**(session 1) (session 2)**

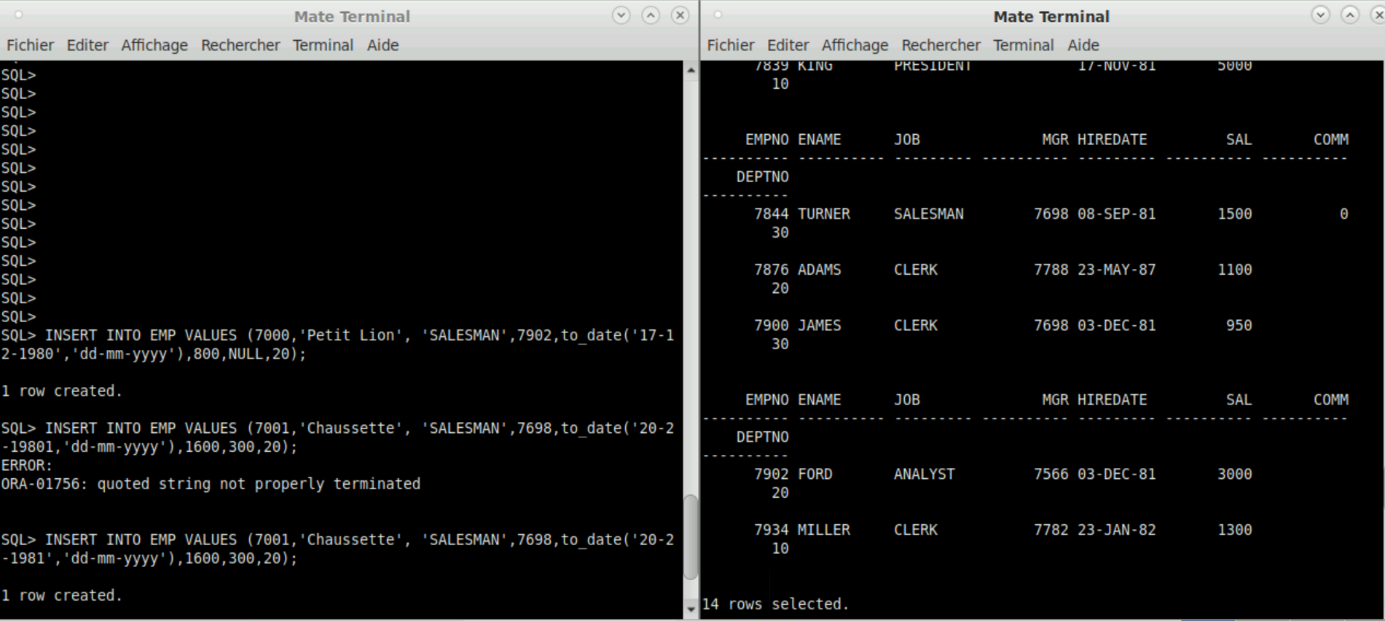


2) Exécution des requêtes SQL à partir de la session 1, création des lignes dans la table EMP.



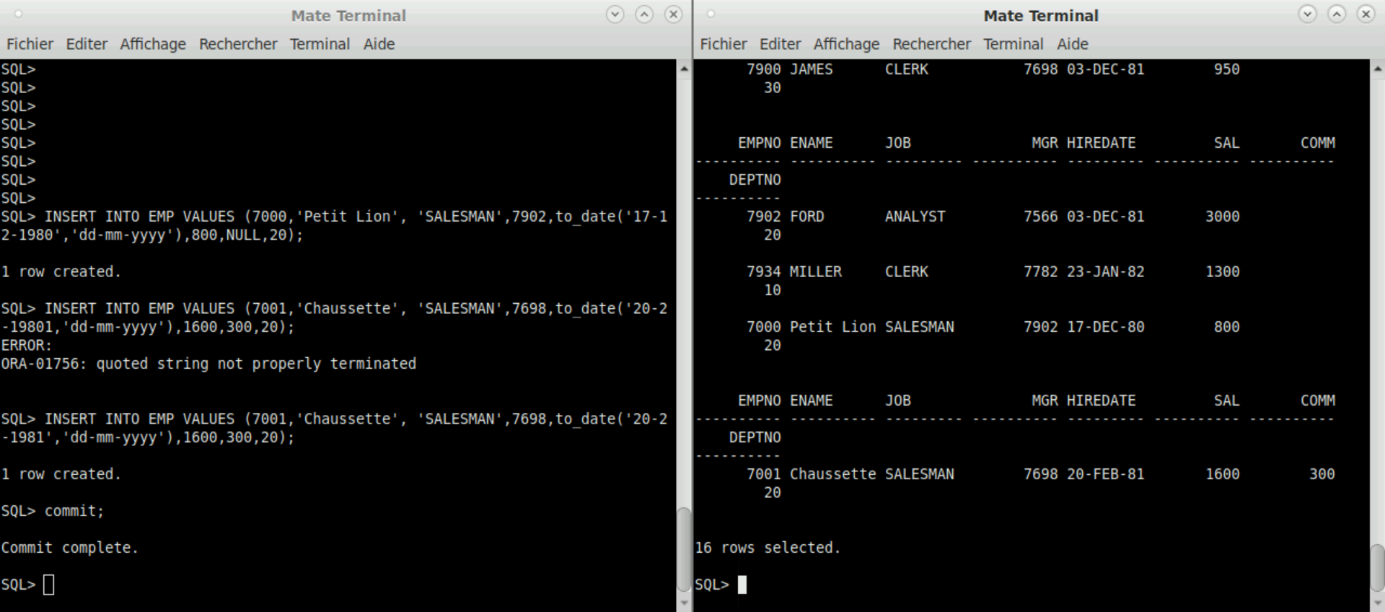
3) Dans la session 2, les modifications apportées précédemment ne sont pas visibles.

**(session1) (session 2)**

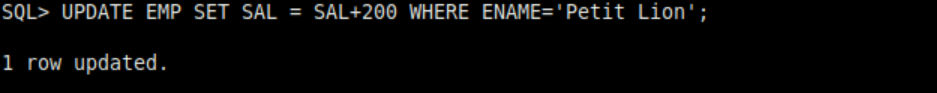


4) Après avoir fait un COMMIT sur la session 1, les modifications apparaissent dans la base de données de la session 2.

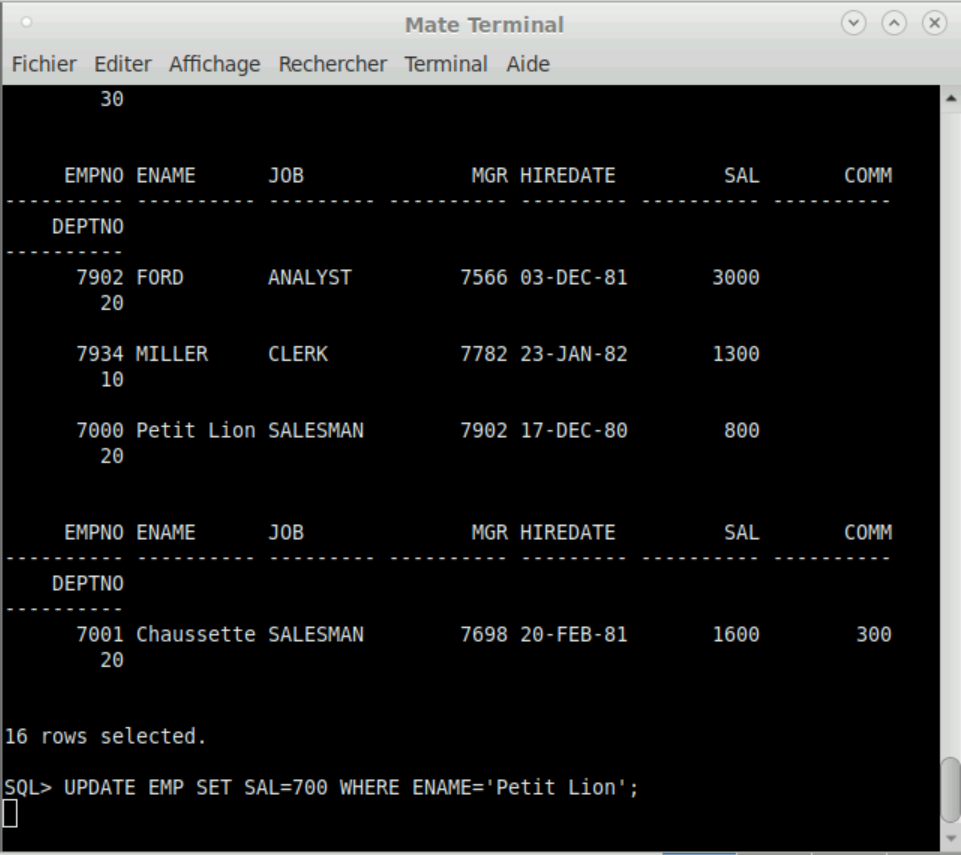
**(session 1) (session 2)**



5)

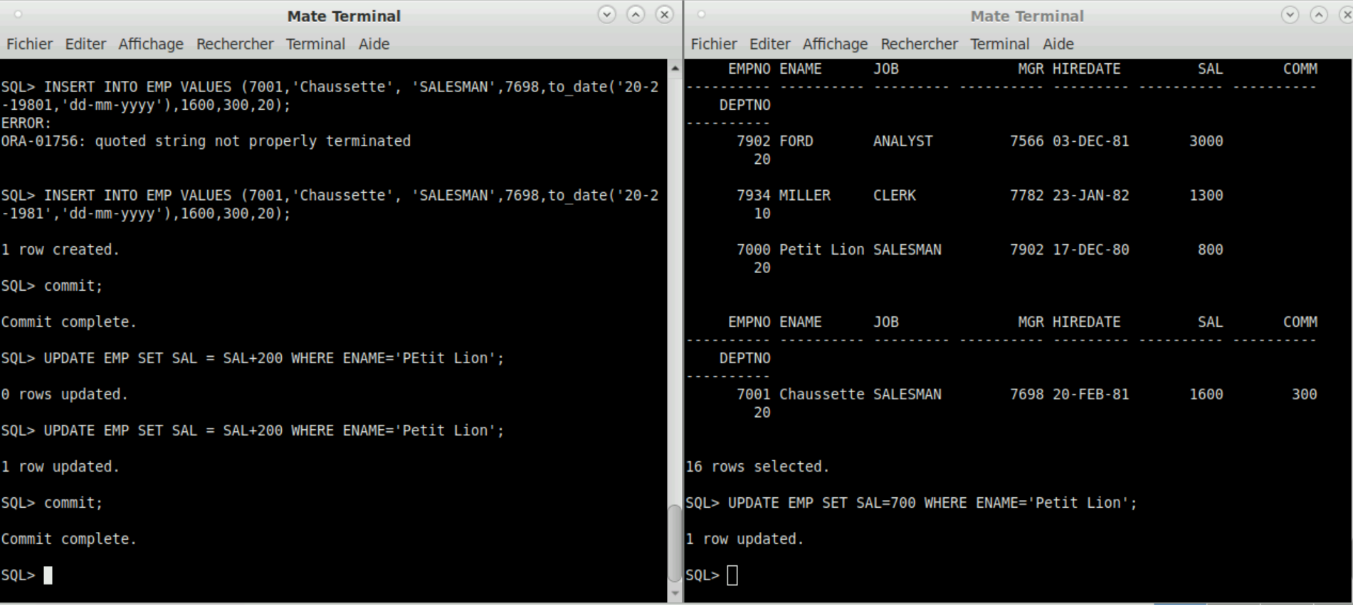


6) En faisant cette requête, le SGBD se met en état bloquant dans la session 2.



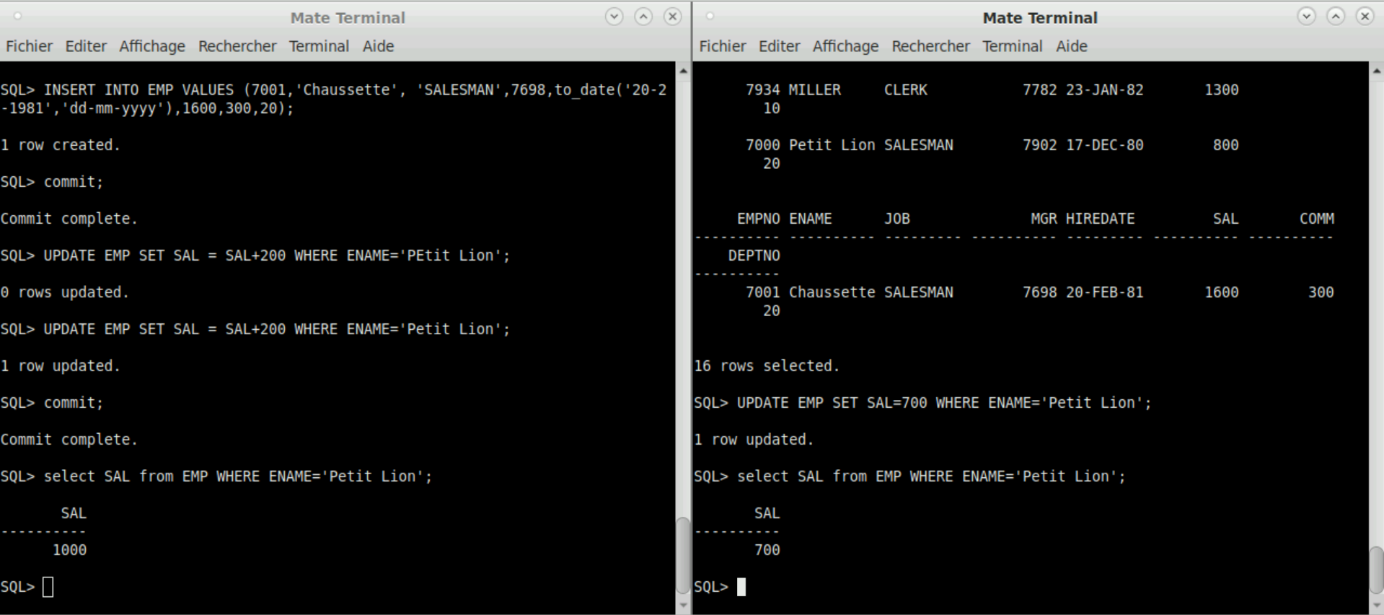
7) Après le COMMIT dans la session 1, la session 2 arrive à faire le UPDATE.

**(session 1) (session 2)**



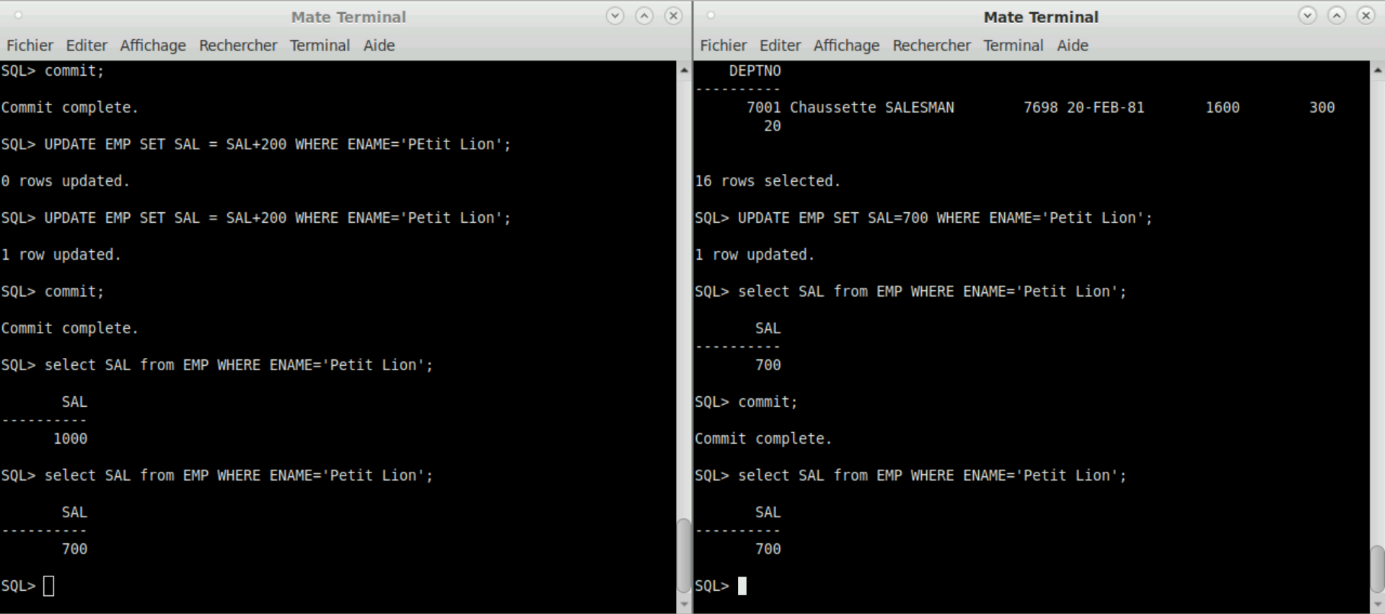
Or quand on fait un SELECT du salaire de ‘Petit Lion’ dans la session  2, on le salaire à bien été changer mais quand on fait la même requête dans la session 1, l’UPDATE de la session 2 n’est pas encore effectif dans la session 1.

**(session 1) (session 2)**



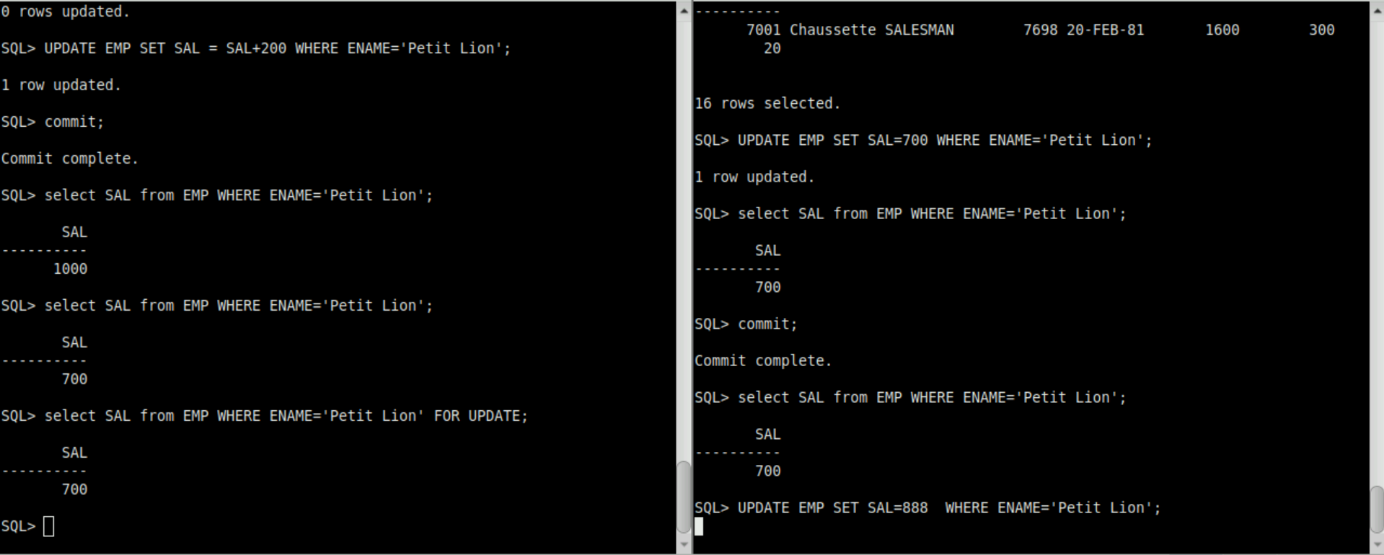
8) Après le COMMIT dans la session 2, l’UPDATE de la session 2 écrase l’UPDATE présente dans la session 1 et met bien à jour sa valeur.

**(session 1) (session 2)**

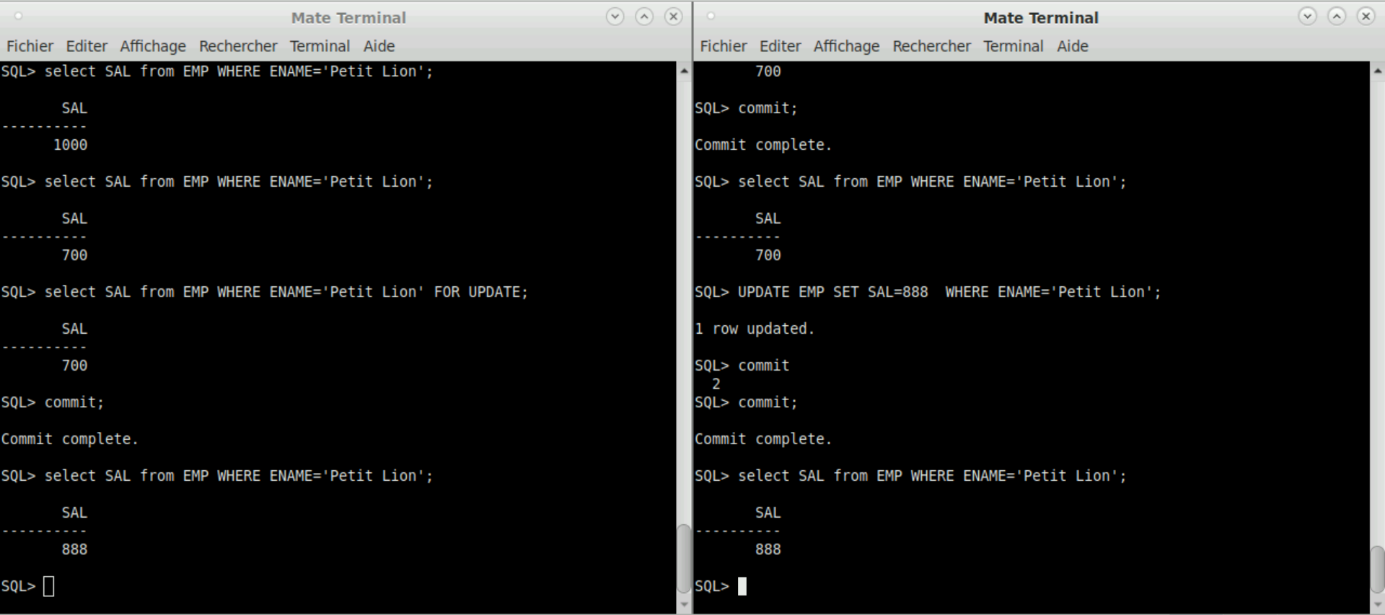


9) Le SELECT FOR UPDATE va se réserver l’accès aux lignes que l'on a sélectionné à partir de la session 1, c’est-à-dire que personne ne pourra modifier les valeurs tant qu’il n’y a pas de COMMIT ou ABORT. Donc avec la session 2, quand on va vouloir faire un UPDATE sur la même ligne ciblée par la session 1 (avec le SELECT FOR UPDATE), la requête SQL va se mettre en état bloquant (ligne ‘Petit Lion’ pour l’exemple).

**(session 1) (session 2)**



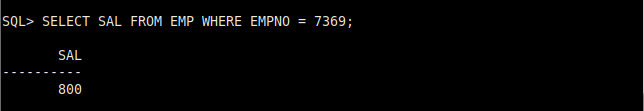
Quand on fait un COMMIT à partir de la session 1, l’UPDATE que l’on a tenté de faire à partir de la session 2 va prendre effet sur la session 2. Ensuite, après un COMMIT sur la session 2, l’UPDATE va prendre effet sur les deux sessions.



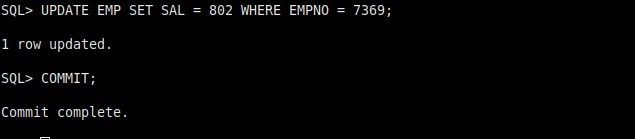
Exercice 2 :

**Scénario 1 :**

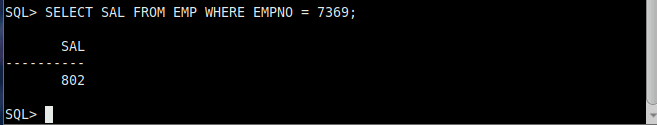
**(session 1)**



**(session 2)**



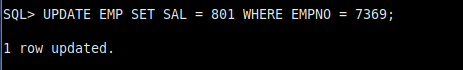
**(session 1)**



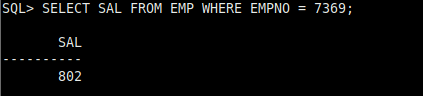
Dans un premier temps, on demande le salaire de l’employé n°7369 qui est 800. Avec la session 2 on définit le salaire de ce même employé à 802 puis on COMMIT sur cette même session. En revenant à la session 1 et en affichant le salaire de cet employé, nous nous rendons compte que son salaire est de 802, donc que la modification de la session 2 est bel et bien visible. Ceci grâce au COMMIT effectué sur la session 2 avant le SELECT de la session 1.

**Scénario 2 :**

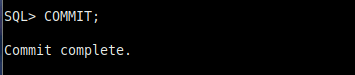
**(session 1)**



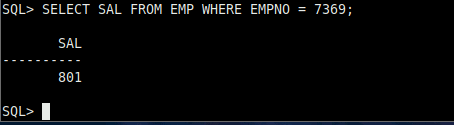
**(session 2)**



**(session 1)**



**(session 2)**

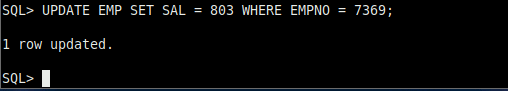


On fait un UPDATE sur le salaire de l’employé n°7369 avec la session 1, puis on regarder le salaire de ce même employé en session 2, et on remarque que la mise à jour de la valeur n’est pas effective (en session 2). On fait par la suite un commit en session 1 et réaffichons le salaire de l’employé en session 2 et là, la valeur est bien mise à jour.

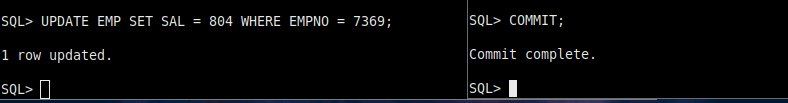
Ce phénomène est possible grâce au COMMIT effectué en session 1 après la modification de valeur du salaire.

**Scénario 3 :**

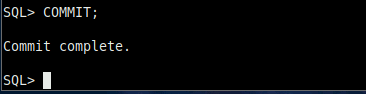
**(session 1)**



**(session 2) -> (session 1)**



**(session 2)**



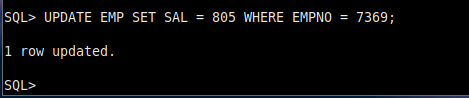
La session 1 met à jour le salaire de l’employé n°7369 à 803.

La session 2 met à jour le salaire de l’employé n°7369 à 804. La ligne est en attente car cette valeur est en cours de modification sur une autre session.

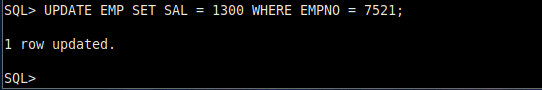
Puis on fait un COMMIT en session 1 puis un COMMIT en session 2. Les deux sessions vont donc modifier le salaire d’origine qui est de 801. La session 1 est la première à COMMIT, le salaire sera alors de 803 puis viens le COMMIT de la session 2 et modifie la valeur qui passe de 803 en 804.

**Scénario 4 :**

**(session 1)**



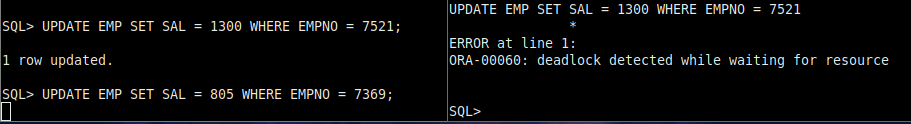
**(session 2)**



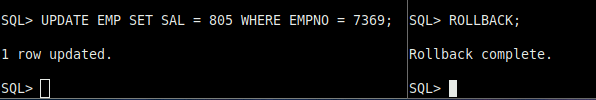
**(session 1)**



**(session 2) (attente)**



**(session 1)**



Ici, les deux sessions vont se bloquées car elles essaient de modifier les mêmes valeurs.

La session 1 veut modifier le salaire de l’employé n°7369 et la session 2 le salaire de l’employé n°7521. Jusqu’ici chaque session attende un COMMIT pour bien terminer leurs transactions. Cependant, la session 1 va essayer à son tour de modifier le salaire de l’employé n°7521 et la session 2 le salaire de l’employé n°7369, les deux sessions se bloquent et l’erreur DEADLOCK est renvoyée. La session 1 effectue un ROLLBACK et reviens aux dernières valeurs enregistrées dans le journal avant la mise à jour. La session 2 est donc débloquée et peux effectuer ses mises à jour.

Exercice 3 :

* Empêcher les lectures non reproductibles :

- Si l’utilisateur ne modifie aucune donnée :

Il suffit de la mettre en mode “read only” avec la commande “SET transaction READ ONLY”.

- Si l’utilisateur modifie des données :

On passe en mode serializable avec la commande “SET transaction isolation level SERIALIZABLE”.

* Empêcher les lignes fantômes :

- Si l’utilisateur ne modifie aucune donnée :

On peut mettre la transaction en read only aussi.

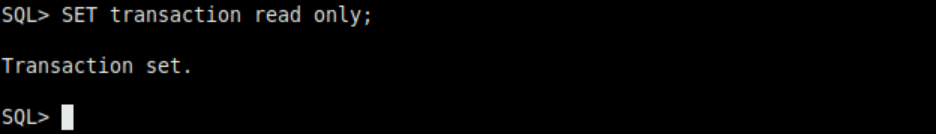
- Si l’utilisateur modifie des données :

On peut bloquer les tables ou se mettre en mode serializable.

**Read Only :**

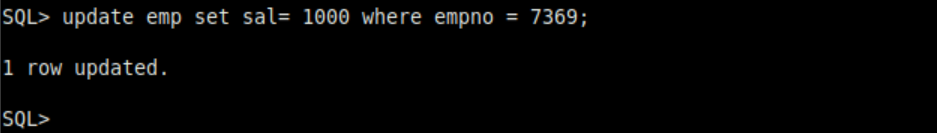
1)

**(session 1)**



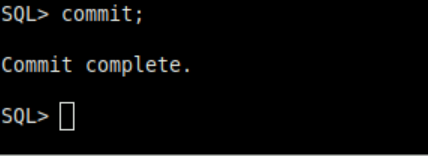
2)

**(session 2)**

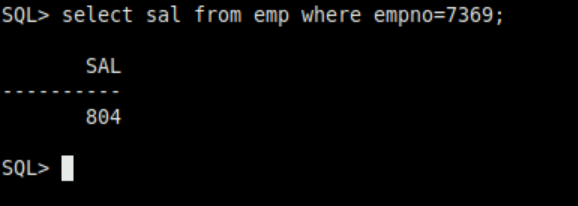


3)

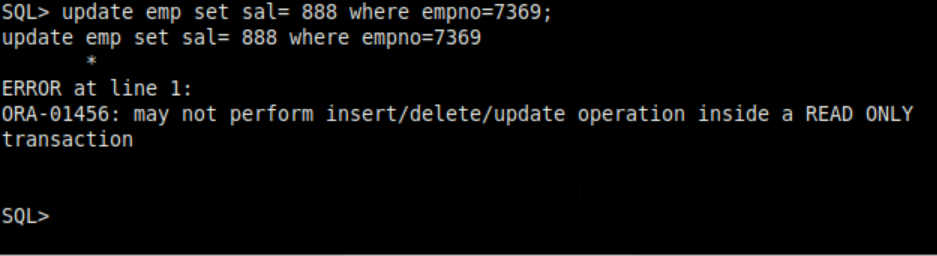
**(session 2)**



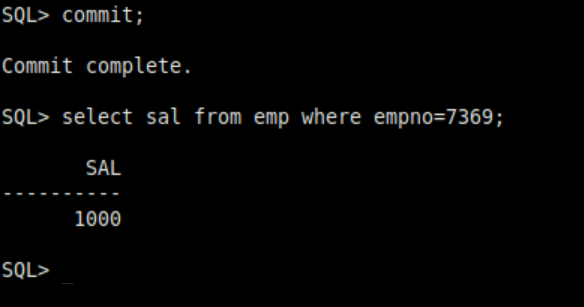
Non on ne voit pas les modifications depuis la 1ere session :



4) Ce n’est pas possible car nous sommes en Read Only dans la session 1 (depuis la question 1).



5) Si le mode “Read Only” n’avait pas été activé dans la première session, on aurait pu voir les données modifiées après le COMMIT fait sur la deuxième session.

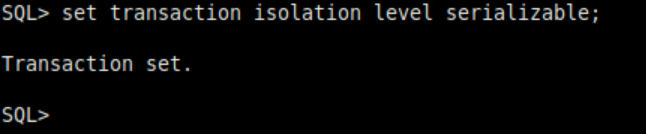


On voit ici qu'après le COMMIT sur la session 1 (*aussi transaction 1*), on peut voir les modifications faites en session 2 précédemment.

**Mode sérialisable**

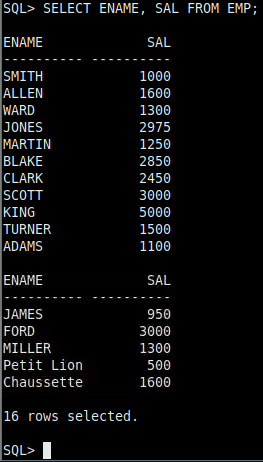
1)2)

**(T1)**



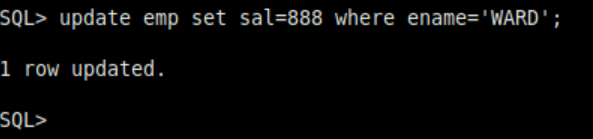
3)

**(T1)**



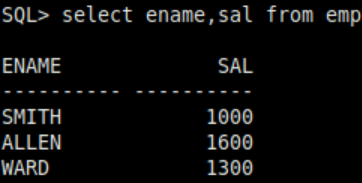
4)

**(T2)**



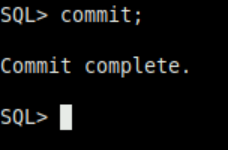
5) On n’a pas encore COMMIT l’update fait dans T2 donc c’est logique que l’on ne voit pas la modification faite sur l’employé “WARD”.

**(T1)**



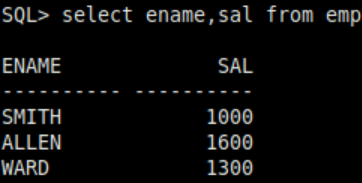
6)

**(T2)**



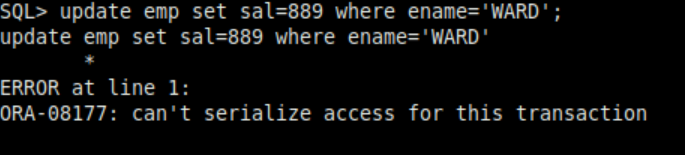
7) On ne voit toujours pas la modification à partir de T1 même après le COMMIT fait en T2, car on a isolé en mode sérialisable T1 au tout début.

**(T1)**



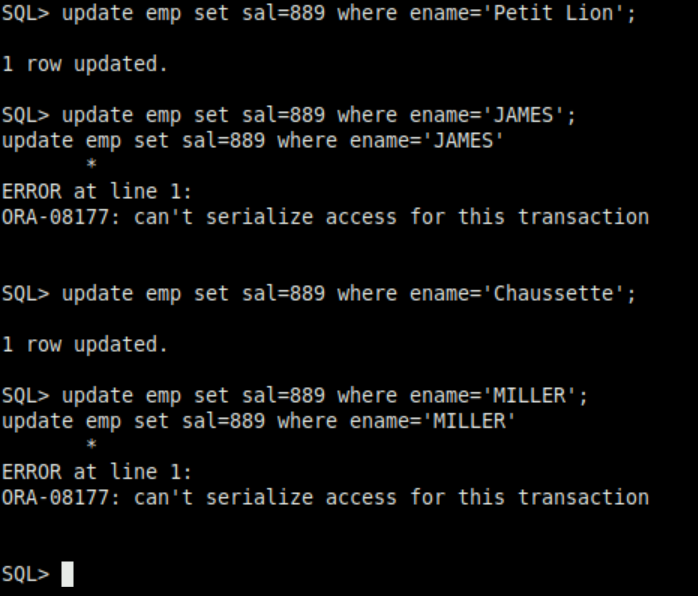
8) On ne peut pas modifier la valeur du salaire de l’employé car on a mis le mode sérialisable sur T1 dès le début.

(T1)



9) Même problème, si l’on veut modifier des valeurs sur la table EMP, on doit faire un COMMIT sur T1. Cependant la modification de valeurs entrées sur cette section oracle ne pose aucun problème (cf. Petit Lion et Chaussette).

**(T1)**



Avec le mode par défaut d’Oracle, T1 aurait vu la modification faite par T2 après le COMMIT. T1 aurait aussi pu modifier le salaire de cet employé.