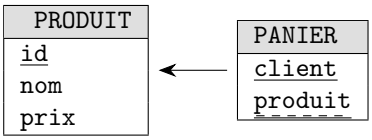
Bases de données avancées

TP4 *Transactions dans Postgres*

Schéma de la base de données



Exercices

Avant de commencer, installer Postgres sur votre machine et connectez-vous au serveur de Postgres.

1. Installer Postgres avec : `sudo apt install postgresql`
2. Lancer le serveur Postgres (en tant que service) : `sudo systemctl start postgresql`
3. Se connecter au serveur via le client (ouvrir une session) : `sudo -u postgres psql`
4. Créer une nouvelle base de données **CREATE** DATABASE tp4;
5. Se connecter à cette nouvelle base `\c tp4`
6. Désactiver **autocommit** dans le client avec (faire attention au majuscule) : `\set AUTOCOMMIT off`
7. Depuis un autre terminal, remplir la base de données avec le fichier `create.sql` : `sudo -u postgres psql tp4 < create.sql`

Par la suite, on travaillera toujours avec deux sessions (terminaux) en parallèle où l'autocommit est désactivé. Vous pouvez observer les verrous qui s'applique à une transaction avec l'extension `pgrowlocks` :

— Il faut l'initialiser avec : **CREATE** EXTENSION IF **NOT EXISTS** pgrowlocks;

— On observe les verrous sur une table avec une commande de la forme **SELECT** ∗ **FROM** pgrowlocks('produit'); où `locked_row` correspond à l'identifiant "physique" du tuple dans Postgres accessible avec la requête **SELECT** ctid, ∗ **from** produit; dans la colonne `ctid`.

— Plus d'information ici : `https://www.postgresql.org/docs/current/pgrowlocks.html`.

Exercice 1. Strict 2PL à la main

Dans cet exercice, il s'agit de suivre un scénario par question en respectant l'ordre des opérations tant que cela est possible : si une session est bloquée, continuer avec les opérations dans l'autre session. On utilisera une seule transaction par session.

Poser des verrous en suivant le protocole 2PL :

— Poser un verrou sur les lignes concernant par une opération avec la commande : **SELECT** ctid, ∗ **FROM** <**table**> **WHERE** <sélection des lignes> FOR <type de verrou> où le type de verrou est soit exclusif **UPDATE** ou partagé SHARE,

— pour détails sur les différents types de verrou explicite : `https://docs.postgresql.fr/16/explicit-locking.html#LOCKING-ROWS`

Question 1. **Lecture concurrente**

1. **Session 1** : Lire les tuples de la table `produit`
2. **Session 2** : Lire le prix des bananes depuis la table `produit`
3. **Session 1** : Valider

4. **Session 2** : Valider

Question 2. **Lecture non répétable**

1. **Session 1** : Lire le tuple de la table `produit` concernant les pommes
2. **Session 2** : Changer le prix des pommes pour 6
3. **Session 1** : Lire à nouveau le tuple concernant les pommes
4. **Session 2** : Annuler
5. **Session 1** : Annuler

Question 3. **Lecture sale**

1. **Session 1** : Changer le prix des pommes pour 6
2. **Session 2** : Lire le tuple de la table `produit` concernant les pommes
3. **Session 1** : Annuler
4. **Session 2** : Annuler

Question 4. **Écriture sale**

1. **Session 1** : Changer le prix des carottes pour 1
2. **Session 2** : Changer le prix des carottes pour 3
3. **Session 1** : Annuler
4. **Session 2** : Annuler

Question 5. **Deadlock**

1. **Session 1** : Changer le prix des pommes pour 4
2. **Session 2** : Changer le prix des carottes pour 3
3. **Session 1** : Changer le prix des carottes pour 1
4. **Session 2** : Changer le prix des pommes pour 6
5. **Session 1** : Annuler
6. **Session 2** : Annuler

Question 6. Proposer un scénario menant à un deadlock comprenant deux lectures et deux mises à jour.

Exercice 2. Tuples fantômes et limites de 2PL

Dans cet exercice, il s'agit de se familiariser avec la notion de tuple fantôme et les problèmes que peuvent poser les conditions de filtres pour les lectures répétées.

Question 7. En posant des verrous suivant le protocole 2PL, suivre le scénario suivant :

1. **Session 1** : Afficher les produits dont le prix supérieur ou égal à 5
2. **Session 2** : Insérer un produit kaki avec le prix 6
3. **Session 2** : Valider
4. **Session 1** : Afficher à nouveau les produits dont le prix supérieur ou égal à 5
5. **Session 1** : Valider

Question 8. Est-ce que l'exécution que vous observez est sérialisable ? Sinon décrire le problème.

Question 9. Montrer que le problème n'est pas spécifique à l'insertion en proposant un scénario avec une lecture non reproductible avec une mise à jour.

Question 10. Considérer un scénario plus complexe sans tuple fantôme : Dans les deux sessions, regarder s'il y a un légume dans le panier du client 0. Si c'est le cas, ajouter un légume différent dans chaque session et valider, sinon annuler.
Est-ce sérialisable ? Peut éviter ce problème avec des verrous sur les lignes ?

Question 11. Proposer et tester une résolution pour les problèmes précédents en utilisant des verrous explicites au niveau des tables :

— `https://docs.postgresql.fr/16/sql-lock.html`
— `https://docs.postgresql.fr/16/explicit-locking.html#LOCKING-TABLES`

Exercice 3. **Le niveau d'isolation** `READ COMMITTED`

Dans cet exercice, nous allons observer le comportement de Postgres dans son niveau d'isolation par défaut `READ COMMITTED`. Pour rappel dans ce niveau d'isolation, une requête ne voit que l'état validée de la base **au début de son éxecution**, plus les mises à jour effectuées dans la transaction locale. Seules les modifications de données nécessitent des verrous exclusifs.

Question 12. En utilisant vos connaissances prévoir ce qu'il se passe lors que l'on essaie d'effectuer une lecture sale dans ce niveau d'isolation. Vérifier ce comportement en pratique avec des requêtes de votre choix.

Question 13. Faire de même pour le cas d'une écriture sale. Ne pas oublier de regarder l'état des verrous dans les différentes sessions.

Question 14. Faire de même pour une lecture non reproductible.

Question 15. Faire apparaitre un ou plusieurs tuples fantômes.

Question 16. Essayer de provoquer un interblocage avec des mises à jour croisées. Est-il possible de provoquer une erreur dans une transaction autrement dans ce niveau d'isolation ?

Question 17. En regardant précisément ce qui se passe au niveau des verrous après les différentes requêtes suivantes, essayer d'expliquer ce qui peut se passer :

1. **Session 1** : Augmenter tous les prix des produits de 1
2. **Session 2** : Supprimer tous les produits dont le prix est 5
3. **Session 1** : Valider
4. **Session 2** : Valider

Si vous ne comprenez pas ou ne vous n'êtes pas sûr de vous, lisez cette section : `https://docs.postgresql.fr/current/transaction-iso.html#XACT-READ-COMMITTED`

Exercice 4. **Le niveau d'isolation** `REPEATABLE READ`

Dans cet exercice, on essaie de comprendre le niveau d'isolation `REPEATABLE READ`, qui est un niveau intermédiaire entre `READ COMMITTED` et `SERIALIZABLE`. Pour rappel dans ce niveau d'isolation, une requête ne voit que l'état validée de la base **au début de la transaction** (au début l'exécution de la première requête, plus précisément), plus les mises à jour effectuées dans la transaction locale.

Question 18. La plupart des phénomènes non sérialisables observés dans le niveau `READ COMMITTED` ne sont plus possibles. Vérifier que cela est bien le cas en pratique.

Question 19. Quels sont les cas précédents provoquant des erreurs ? Y a-t-il d'avantage d'erreur dans les transactions avec le niveau d'isolation `REPEATABLE READ` en comparaison avec celui `READ COMMITTED` ?

Question 20. Il y a encore des cas où il y a des anomalies de sérialisabilité apparaîssent dans ce niveau d'isolation. Par exemple, vérifier qu'il y en a un sur le scénario suivant :

— On débute avec un client a des pommes et des poires dans son panier.
— Dans chacune des deux sessions, regarder s'il reste au moins deux fruits dans son panier,
— si c'est le cas, supprimer un fruit du panier (différent dans chaque session)
— sinon annuler.

Question 21. Vérifier que le phénonème précédent est bien évité avec le niveau d'isolation `SERIALIZABLE`.

Exercice 5. **Clés étrangères et verrous** Dans cet exercice, on souhaite comprendre la différence entre les quatres verrous explicites au niveau des lignes utilisés par Postgres : `https://docs.postgresql.fr/16/explicit-locking.html#LOCKING-ROWS`.

Question 22. Observer les verrous des deux tables lorsqu'une ligne de la table `panier` est modifiée. Que remarquez-vous ?

Question 23. Est-il possible de modifier le prix d'un produit et de manière concurrente une entrée dans un panier qui référence ce même produit ? Pourquoi ?

Question 24. Est-il possible de supprimer un produit et de manière concurrente une entrée dans un panier qui référence ce même produit ? Pourquoi ?

Question 25. Est-il possible de modifier l'identifiant d'un produit et de manière concurrente une entrée dans un panier qui référence ce même produit ? Pourquoi ?