Commencé le	mercredi 1 avril 2020, 13:34
État	Terminé
Terminé le	jeudi 2 avril 2020, 13:29
Temps mis	23 heures 54 min
Note	**20,00** sur 20,00 (**100**%)

Description

Voici une série de situations rencontrées lors du développement d'une architecture de base de données. Il est fortement encouragé de s'appuyer sur la documentation officielle de Postgres 11 (lien dans le sujet de TP) et de tester le code des questions sur la machine virtuelle (ouvrez plusieurs consoles SSH pour tester les transactions parallèles).

Pour les questions à choix multiples, il faut cocher **le ou les bonnes réponses**

Question **1**

Correct

Note de 2,50 sur 2,50

Pour comprendre le principe de transaction, Les commandes suivantes sont exécutées :

Console 1	Console 2	Console 3
postgres=# SELECT now();	postgres=# SELECT now(), now();	BEGIN;
postgres=# SELECT now();		postgres=# SELECT now();
		postgres=# SELECT now();

Considérez les affirmations suivantes :

Veuillez choisir au moins une réponse :

- [x] a. les deux temps renvoyés par console 3 vont être identiques ✔
- [x] b. les deux temps renvoyés par console 2 vont être identiques ✔
- [ ] c. La fonction now() en SQL renvoie le temps processeur à chaque exécution
- [ ] d. les deux temps renvoyés par console 1 vont être identiques

Votre réponse est correcte.

Les réponses correctes sont : les deux temps renvoyés par console 2 vont être identiques, les deux temps renvoyés par console 3 vont être identiques

Question **2**

Correct

Note de 2,50 sur 2,50

Considérer les deux transactions suivantes :

Transaction 1	Transaction 2

test=# BEGIN;
test=# SELECT 4;
test=# SELECT 4 / 0;
test=# SELECT 4;
test=# COMMIT;
	test=# BEGIN;
	test=# SELECT 4;
	test=# SAVEPOINT svpt;
	test=# SELECT 4 / 0;
	test=# SELECT 4;
	test=# ROLLBACK TO SAVEPOINT svpt;
	test=# SELECT 5;

Considérez les affirmations suivantes :

Veuillez choisir au moins une réponse :

- [ ] a. SAVEPOINT permet de restaurer l'état de la table après un COMMIT;
- [x] b. ROLLBACK TO SAVEPOINT permet de revenir à un point de la transaction en cours pour éviter que toutes les requêtes de la transaction ne soient annulées ✔
- [ ] c. RELEASE SAVEPOINT permet aussi de revenir à un point de la transaction en cours pour éviter que toutes les requêtes de la transaction ne soient annulées
- [x] d. Postgres va ignorer toutes requêtes subséquentes à une erreur provoquée (division par zero par exemple) par une requête au sein d'une transaction. ✔

Votre réponse est correcte.

Les réponses correctes sont : Postgres va ignorer toutes requêtes subséquentes à une erreur provoquée (division par zero par exemple) par une requête au sein d'une transaction., ROLLBACK TO SAVEPOINT permet de revenir à un point de la transaction en cours pour éviter que toutes les requêtes de la transaction ne soient annulées

Question **3**

Correct

Note de 2,50 sur 2,50

Les options de Postgres sont laissées à défaut. Les commandes suivantes sont exécutées :

postgres=#CREATE DATABASE bank;

postgres=# \connect bank;

bank=# CREATE TABLE operations (id int, amount float, PRIMARY KEY (id));

bank=# INSERT INTO operations VALUES (1,-100);

bank=# INSERT INTO operations VALUES (2,+200);

bank=# INSERT INTO operations VALUES (3,-10.2);

Les deux transactions sont exécutées en parallèle :

Temps	Transaction 1 (console 1)	Transaction 2 (console 2)
1	bank=# BEGIN;	
2	bank=# SELECT sum(amount) FROM operations; sum ------ 89.8	
3		bank=# BEGIN;
4		bank=# INSERT into operations VALUES (4,300);
5		bank=# COMMIT;
6	bank=# SELECT sum(amount) FROM operations;	

x

Quel est le retour de la requête 6 (solde du compte) ?

Réponse : 389.8 ✔

Résultat : 389.8

En effet, de base postgres fonctionne en mode READ COMMITED; A chaque requête, un image de la table est reprise.

La réponse correcte est : 389,8

Question 4

Correct

Note de 2,50 sur 2,50

Les commandes suivantes sont exécutées :

postgres=#CREATE DATABASE bank;

postgres=# \connect bank;

bank=# CREATE TABLE operations2 (id int, amount float, PRIMARY KEY (id));

bank=# INSERT INTO operations2 VALUES (1,-100);

bank=# INSERT INTO operations2 VALUES (2,+200);

bank=# INSERT INTO operations2 VALUES (3,-10.2);

Les deux transactions sont exécutées en parallèle :

Temps	Transaction 1 (console 1)	Transaction 2 (console 2)
1	bank=# BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;	
2	bank=# SELECT sum(amount) FROM operations2; sum ------ 89.8	
3		bank=# BEGIN;
4		bank=# INSERT into operations2 VALUES (4,150);
5		bank=# COMMIT;
6	bank=# SELECT sum(amount) FROM operations2;	

x

Quel est le retour de la requête 6 (solde du compte) ?

Réponse : 89.8 ✔

Résultat : 89.8

En effet, en mode REPETABLE READ, une image de labase actuelle est prise au début de la transaction.

La réponse correcte est : 89,8

Question **5**

Correct

Note de 2,50 sur 2,50

Les commandes suivantes sont exécutées :

postgres=#CREATE DATABASE bank;

postgres=# \connect bank;

bank=# CREATE TABLE operations3 (id int, amount float, PRIMARY KEY (id));

bank=# INSERT INTO operations3 VALUES (1,-100);

bank=# INSERT INTO operations3 VALUES (2,+200);

bank=# INSERT INTO operations3 VALUES (3,-10.2);

Les deux transactions sont exécutées en parallèle :

Temps	Transaction 1 (console 1)	Transaction 2 (console 2)
1	bank=# BEGIN;	bank=# BEGIN;
2	bank=# UPDATE operations3 SET amount=-11 WHERE id = 3;	bank=# UPDATE operations3 SET amount=-12 WHERE id = 3;
3	bank=#COMMIT;	bank=#COMMIT;
4	bank=# SELECT * FROM operations3; id | amount ----+-------- 1 | -100 2 | 200 3 | -12 (3 lignes)	

Pourquoi la ligne 4 affiche -12 sur la console 1 ? Comment y remédier ?

Veuillez choisir au moins une réponse :

- [x] a. Il faut commencer la seconde transaction par : begin transaction isolation level serializable; pour éviter l'écrasement de la première transaction ✔
- [ ] b. Il faut commencer la seconde transaction par : begin transaction isolation repeatable read ; pour éviter l'écrasement de la première transaction
- [x] c. Les deux transactions sont concurrentes et la seconde transaction à écrasé les modifications de la première ✔
- [ ] d. Cette situation ne risque pas de causer de corruption de données grâce aux transactions utilisées

Votre réponse est correcte.

Les réponses correctes sont : Les deux transactions sont concurrentes et la seconde transaction à écrasé les modifications de la première, Il faut commencer la seconde transaction par : begin transaction isolation level serializable; pour éviter l'écrasement de la première transaction

Question **6**

Correct

Note de 2,50 sur 2,50

Les commandes suivantes sont exécutées :

postgres=#CREATE DATABASE bank;

postgres=# \connect bank;

bank=# CREATE TABLE user (id int);

bank=# INSERT INTO user VALUES (1), (2);

Les deux transactions sont exécutées en parallèle :

Temps	Transaction 1 (console 1)	Transaction 2 (console 2)
1	bank=# BEGIN;	bank=# BEGIN;
2	bank=# UPDATE user SET id = id * 5 WHERE id = 1;	bank=# UPDATE user SET id = id * 5 WHERE id = 2;
3	bank=# UPDATE user SET id = id * 10 WHERE id = 2;	
4		bank=# UPDATE user SET id = id * 10 WHERE id = 1;

Que peut-on conclure ?

Veuillez choisir au moins une réponse :

- [x] a. Deux transactions attendent l'une après l'autre, on a donc un blocage mortel (deadlock) ✔
- [x] b. Si les requête 2 et 4 sont transformées en SELECT, il n'y a plus de problème ✔
- [ ] c. Si la requête 2 est transformée en DELETE, il n'y a plus de problème
- [ ] d. Si la requête 4 est transformée en DELETE, il n'y a plus de problème

Votre réponse est correcte.

Les réponses correctes sont : Deux transactions attendent l'une après l'autre, on a donc un blocage mortel (deadlock), Si les requête 2 et 4 sont transformées en SELECT, il n'y a plus de problème

Question 7

Correct

Note de 2,50 sur 2,50

Les commandes suivantes sont exécutées :

postgres=#CREATE DATABASE booking;

postgres=# \connect booking;

booking=# CREATE TABLE siege (id int);

booking=# INSERT INTO siege VALUES (1),(2),(3),(4),(5),(6),(7),(8),(9);

Les deux transactions sont exécutées en parallèle :

Temps	Transaction 1 (console 1)	Transaction 2 (console 2)
1	booking=# BEGIN;	booking=# BEGIN;
2	booking=# SELECT * FROM siege LIMIT 1 FOR UPDATE;	booking=# SELECT * FROM siege LIMIT 1 FOR UPDATE;

Imaginez que ce système est utilisé pour réserver les sièges d'un avion, le fonctionnement est-il satisfaisant ? comment y remédier ?

Veuillez choisir au moins une réponse :

- a. SELECT FOR KEY SHARE; permet à plusieurs clients de réserver un siège en même temps en garantissant l'absence de transactions concurrentes.
- b. Oui, cela permet à plusieurs personnes de réserver un siège en même temps.
- c. Non, ce n'est pas satisfaisant, un seul client peut réserver un siège à la fois ✔
- d. SELECT FOR UPDATE SKIP LOCKED; permet à plusieurs clients de réserver un siège en même temps en garantissant l'absence de transactions concurrentes. ✔

Votre réponse est correcte.

Les réponses correctes sont : Non, ce n'est pas satisfaisant, un seul client peut réserver un siège à la fois, SELECT FOR UPDATE SKIP LOCKED; permet à plusieurs clients de réserver un siège en même temps en garantissant l'absence de transactions concurrentes.

Question **8**

Correct

Note de 2,50 sur 2,50

Les commandes suivantes sont exécutées :

postgres=#CREATE DATABASE outils;

postgres=# \connect outils;

outils=# CREATE TABLE tournevis (id int, name varchar(255));

outils=# CREATE TABLE vis (id int, name varchar(255));

outils=# INSERT INTO tournevis VALUES (1,'cruciforme');

outils=# INSERT INTO tournevis VALUES (2,'plat');

Les deux transactions sont exécutées en parallèle :

Temps	Transaction 1 (console 1)	Transaction 2 (console 2)
1	outils=# BEGIN;	outils=# BEGIN;
2	outils=# SELECT * FROM tournevis FOR UPDATE;	outils=# UPDATE tournevis SET name='carre' WHERE id = 2;

Considérer les affirmations suivantes :

Veuillez choisir au moins une réponse :

- [x] a. SELECT FOR UPDATE permet de bloquer l'écriture d'autres transactions pendant une transaction donnée. ✔
- [ ] b. Si la première transaction effectue une requête update, La seconde transaction voit les modifications de la première transaction, même si la première transaction n'a pas COMMIT.
- [ ] c. SELECT FOR UPDATE permet de bloquer la lecture d'autres transactions pendant une transaction donnée.
- [x] d. La seconde transaction va attendre que la première transaction soit terminée ✔

Votre réponse est correcte.

Les réponses correctes sont : SELECT FOR UPDATE permet de bloquer l'écriture d'autres transactions pendant une transaction donnée., La seconde transaction va attendre que la première transaction soit terminée

◄ TP5 - Sujet

Aller à...

Lien téléchargement VM ►

/