420-C42

Langages d'exploitation des bases de données

Partie 15

TCL

Contrôle de transaction, contraintes déférées
et variables d’environnement

TCL introduction

- Le terme TCL fait référence à « *Transaction Control Langage* » ou plutôt langage de contrôle de transaction.
- La notion de transaction est fondamentale et très importante. Elle fait référence au fait qu'un groupe d'opérations constitue un tout indissociable.
- Ce concept précise que :
  - l'ensemble des opérations constituant la transaction sont toutes réalisées ou aucune ne l'est (tout ou rien)
  - les étapes intermédiaires de la transaction sont invisibles aux autres transactions réalisées au même moment
  - si une panne survient pendant la transaction, l'utilisateur est informé et le système revient à son état antérieur
  - si la transaction est réalisée avec succès, l'utilisateur est informé, les données deviennent visibles par tous et sont sécurisées (même s'il y a une panne immédiate)

TCL

ACID

- De façon plus formelle, l'acronyme ACID représente les caractéristiques importantes pour qu'une transaction garantisse une exécution viable.
  - **A**tomicité (au sens de la transaction)
    - La transaction se fait au complet ou pas du tout.
  - **C**ohérence
    - La transaction passera d'un état valide à un autre état valide. La validité est définie selon la conception initiale de l'architecture des données.
  - **I**solation
    - La transaction est isolée de toute autre transaction. Elle reste invisible des autres opérations du SGBD tant qu'elle n'est pas confirmée.
  - **D**urabilité
    - Lorsque la transaction est confirmée, elle devient persistante dans la base de données peu importe les pannes potentielles.

TCL

concept

- Le TCL est lié aux opérations du DML.
- L'exemple typique :
  - On désire faire la transaction bancaire suivante :
    Frédérick fait un transfert de 1000$ à Caroline
  - On voit qu'au moins 2 opérations sont nécessaires :
    - retirer 1000$ à Frédérick
    - déposer 1000$ à Caroline.
  - Il est impensable que cette transaction soit faite à moitié. Soit le 1000$ reste à Frédérick soit il est transféré à Caroline. Il ne peut pas être débité sans être déposé. C'est l'usage du TCL qui permet de s'en assurer.

Seul deux états sont acceptables : état initial ou état final

TCL BEGIN & COMMIT

- Avec PostgreSQL, on détermine une transaction en encadrant toutes les opérations contenues entre les commandes BEGIN et COMMIT. On nomme les opérations concernées bloc transactionnel (ou « *transaction bloc* »).
- Chaque opération fait partie d'une transaction. PostgreSQL ajoute implicitement un COMMIT à toutes les opérations ne faisant pas partie d'un bloc transactionnel.

```
-- l'approche suivante est erronée et constitue 2 transactions indépendantes
UPDATE client SET avoir = avoir - 1000 WHERE prenom = 'Frédérick';
UPDATE client SET avoir = avoir + 1000 WHERE prenom = 'Caroline';

-- la solution : une seule transaction
BEGIN;
    UPDATE client SET avoir = avoir - 1000 WHERE prenom = 'Frédérick';
    UPDATE client SET avoir = avoir + 1000 WHERE prenom = 'Caroline';
COMMIT;
```

TCL SAVEPOINT & ROLLBACK

- Il est possible d'arrêter formellement une transaction en cours et revenir à l'état initial. La commande ROLLBACK effectue ce retour.
- Il est aussi possible de définir un point de reprise dans la transaction avec la commande SAVEPOINT *nom*; et d'y revenir avec ROLLBACK TO *nom*;. Cette opération n'annule pas la transaction en cours.

```
-- un exemple
BEGIN;
    UPDATE client SET avoir = avoir - 1000 WHERE prenom = 'Gustave';          -- (1)
    UPDATE client SET avoir = avoir + 1000 WHERE prenom = 'Gaétan';           -- (2)
ROLLBACK;                                  -- Oups, mauvais clients - on annule 1 et 2
BEGIN:
    UPDATE client SET avoir = avoir - 1000 WHERE prenom = 'Frédéric';         -- (3)
    SAVEPOINT mon_point_de_reprise;
    UPDATE client SET avoir = avoir + 1000 WHERE prenom = 'Gustave';          -- (4)
    ROLLBACK TO mon_point_de_reprise;          -- Aouch, mauvais client on annule 4
    UPDATE client SET avoir = avoir + 1000 WHERE prenom = Caroline';          -- (5)
COMMIT;                                  -- au final, seule les opérations 3 et 5 sont réalisées
```

Contraintes différées deferrable et deferred

- Nous avons vu les problèmes associés à la déclaration de tables ayant une dépendance circulaire liée aux opérations du DML (DCDDL) (ou *dépendance structurelle circulaire*).
- La solution est simple, **après** avoir créé toutes les tables, on modifie les tables en ajoutant les contraintes de clés étrangères.
- Néanmoins, il existe aussi le problème de dépendance circulaire liées aux opérations du DML (DCDML) (ou *dépendance de données circulaires*).
- En fait, si les colonnes associées aux contraintes de clé étrangères sont déclarées NOT NULL, il devient impossible d'insérer les valeurs.
- Même si les colonnes concernées autorisent les valeurs nulles, il existe des solutions simples qui restent inélégantes.

Contraintes différées deferrable et deferred

- Solution **inadéquate** 1 : forcer les contraintes de clé étrangères à NULL :
  1. on réalise l'insertion dans une première table en mettant la référence à NULL
  2. on réalise l'insertion dans la deuxième table en mettant la référence vers la valeur attendue de la première table
  3. on modifie la valeur de la première table en effectuant la mise à jour selon la donnée de la deuxième table

Contraintes différées deferrable et deferred

- Solution **inadéquate** 2 : laisser les contraintes de clé étrangères à NOT NULL mais en insérant d'abord une valeur bidon pour les opérations intermédiaires:
  1. on réalise l'insertion dans une première table en mettant la référence à la valeur bidon de la deuxième table
  2. on réalise l'insertion dans la deuxième table en mettant la référence vers la valeur attendue de la première table
  3. on modifie la valeur de la première table en effectuant la mise à jour selon la donnée de la deuxième table

Contraintes différées deferrable et deferred

- Solution **inadéquate** 3 : laisser les contraintes de clé étrangères à NOT NULL mais on suspend temporairement l'application des contraintes de clés étrangères:
  1. on suspend les contraintes sur la table 1 ALTER TABLE xyz DISABLE TRIGGER ALL;
  2. on réalise l'insertion dans une première table en mettant la référence à une valeur quelconque temporaire
  3. on réalise l'insertion dans la deuxième table en mettant la référence vers la valeur attendue de la première table
  4. on modifie la valeur de la première table en effectuant la mise à jour selon la donnée de la deuxième table
  5. on rétabli les contraintes sur la table 1 ALTER TABLE xyz ENABLE TRIGGER ALL;
- Attention, lors de l'étape 5, le système ne valide pas les operations faites aux étapes 2 et 4. L'intégrité des données n'est plus garantie.

Contraintes différées deferrable et deferred

- Solution finale **adéquate** : on utilise la notion de contraintes différées
  - Il est possible de déclarer une contrainte de clé étrangère différable, c'est-à-dire que la validation de la valeur ne sera faite seulement après la transaction.
  - Ainsi, dans une transaction, entre le BEGIN et le COMMIT, il est possible de mettre n'importe quelle valeur et de la modifier avant la fin. Seulement après le COMMIT, la valeur est validée.
  - C'est lors de la création de la clé étrangère qu'on peut déterminer si elle peut être différée (différable).
  - Lorsqu'une contrainte est déclarée différable, elle peut être dans l'un de ces deux états :
    - immédiat : la contrainte sera validée immédiatement, elle ne sera pas différée
    - différé : la contrainte sera validée après la transaction, elle sera différée

Contraintes différées deferrable et deferred

- Solution finale, exemple syntaxique :

```
-- Ajout d'une clé étrangère différable
ALTER TABLE employe
    ADD CONSTRAINT fk_emp_dep
        FOREIGN KEY (dep)
        REFERENCES departement(id)
        DEFERRABLE                  -- ou NOT DEFERRABLE
        INITIALLY DEFERRED;         -- ou INITIALLY IMMEDIATE

-- Si la clé étrangère est différable, il est possible de changer son état explicitement :

SET CONSTRAINTS fk_emp_dep DEFFERED;
SET CONSTRAINTS fk_emp_dep IMMEDIATE; -- effectue la validation instantanément
```

Contraintes différées deferrable et deferred

- Solution finale : étapes :
  1. on s’assure que les clés étrangères soient différables
  2. on s’assure que les clés étrangères soient en mode différée
  3. on démarre une transaction BEGIN
  4. on réalise l’insertion dans une première table en mettant la référence à une valeur quelconque temporaire
  5. on réalise l’insertion dans la deuxième table en mettant la référence vers la valeur attendue de la première table
  6. on modifie la valeur de la première table en effectuant la mise à jour selon la donnée de la deuxième table
  7. on termine la transaction COMMIT
  8. optionnellement et selon le contexte, on peut mettre/remettre les clés étrangères en mode immédiat