Data Control Language

GESTION DES RÔLES ET DROITS

Rôles

Un rôle est
- soit un utilisateur de la BD,
- soit un groupe d'utilisateurs de la BD.

Les rôles peuvent
- posséder des objets de la BD, par exemple des tables et des fonctions,
- affecter des droits sur ces objets à d'autres rôles pour contrôler qui a accès à ces objets.

Rôles

Il est possible de donner l'*appartenance* d'un rôle à un autre rôle, autorisant celui-ci à utiliser les droits affectés à celui-là.

Le concept de rôle englobe celui d'utilisateur et celui de groupe :

Tout rôle peut agir comme un utilisateur, un groupe ou les deux.

Rôle postgres

A l'installation, un système contient toujours un rôle prédéfini.

Ce rôle est un superutilisateur et aura par défaut le même nom que l'utilisateur du système d'exploitation qui a initialisé le groupe de bases de données.

Par habitude, ce rôle est nommé postgres.

Pour créer d'autres rôles :

CREATE ROLE nom ;

Pour cela il faut être connecté en tant que ce rôle initial

Superuser

Les superutilisateurs ne sont pas sujets aux vérifications des droits
- Ceci est un droit dangereux et ne devrait pas être utilisé sans faire particulièrement attention
- Il est préférable de travailler autant que possible avec un rôle qui n'est pas superutilisateur

Pour créer un nouveau superutilisateur :

CREATE ROLE *nom_utilisateur* SUPERUSER;

On ne peut faire cela qu'en tant que superutilisateur

Attributs des rôles

- Droit de connexion
- Droit de création de BD
- Droit de création de rôle

Droit de connexion

les rôles disposant de l'attribut LOGIN peuvent être utilisés comme rôle initial pour une connexion à une BD.

Création d'un rôle disposant du droit de connexion

CREATE ROLE nom_utilisateur LOGIN;

ou bien

CREATE USER nom_utilisateur;

Droit de création de rôles

Un rôle non superutilisateur doit se voir explicitement donné le droit de créer des rôles :

CREATE ROLE *nom_utilisateur* CREATEROLE;

Un rôle disposant du droit CREATEROLE peut modifier et supprimer d'autres rôles, et donner ou supprimer l'appartenance à ces rôles.

Appartenance d'un rôle

Il est souvent intéressant de grouper les utilisateurs pour faciliter la gestion des droits : de cette façon, les droits peuvent être donnés ou supprimés pour tout un groupe.

Dans PostgreSQL, ceci se fait en créant un rôle représentant le groupe, puis en ajoutant les rôles utilisateurs individuels comme membres de ce groupe.

Droit de création de bases de données

Les droits de création de BD doivent être explicitement donnés à un rôle (à l'exception des superutilisateurs).

Création d'un rôle ayant le droit de création d'une BD:

CREATE ROLE nom_utilisateur CREATEDB;

Configurer un rôle en tant que groupe

CREATE ROLE *nom_groupe*;

GRANT *nom_groupe* TO *role1*, ... ;

REVOKE *nom_groupe* FROM *role1*, ... ;

Sous-groupes possibles : les membres peuvent aussi être des groupes

Utilisation des droits d'un rôle groupe

Les membres d'un rôle groupe peuvent utiliser les droits du rôle de deux façons :

- SET ROLE pour donner temporairement le rôle du groupe.
  Dans cet état, la session de la BD a accès aux droits du rôle groupe plutôt qu'à ceux du rôle de connexion.
- Les rôles membres qui ont l'attribut INHERIT peuvent utiliser automatiquement les droits des rôles dont ils sont membres, ceci incluant les droits hérités par ces rôles.

Utilisation des droits d'un rôle groupe

```
CREATE ROLE joe LOGIN INHERIT;
CREATE ROLE admin NOINHERIT;
CREATE ROLE wheel NOINHERIT;
GRANT admin TO joe;
GRANT wheel TO admin;
```

Après connexion en tant que joe, la session peut utiliser :

- les droits donnés directement à joe
- les droits donnés à admin
- mais pas les droits donnés à wheel

Utilisation des droits d'un rôle groupe

```
CREATE ROLE joe LOGIN INHERIT;
CREATE ROLE admin NOINHERIT;
CREATE ROLE wheel NOINHERIT;
GRANT admin TO joe;
GRANT wheel TO admin;
-- connexion en tant que joe
SET ROLE admin;
```

La session peut utiliser les droits donnés à admin mais n'a plus accès à ceux de joe.

```
SET ROLE wheel;
```

la session peut utiliser uniquement les droits de wheel, mais plus ceux de joe ni ceux de admin.

Utilisation des droits d'un rôle groupe

L'état du droit initial peut être restauré avec une des instructions suivantes:

```
SET ROLE joe;
SET ROLE NONE;
RESET ROLE;
```

Suppression d'un rôle

- Tout objet appartenant à un rôle doit d'abord être supprimé ou réaffecté à d'autres propriétaires
- Tout droit donné à un rôle doit être révoqué.

Ecart à la norme SQL

- Dans la norme, il existe une distinction claire entre utilisateurs et rôles:
  les utilisateurs ne peuvent pas hériter automatiquement alors que les rôles le peuvent.
- Ce comportement est obtenu dans PostgreSQL en donnant aux rôles utilisés comme des rôles SQL l'attribut INHERIT, et en donnant aux rôles utilisés en tant qu'utilisateurs SQL l'attribut NOINHERIT.

Organisation de postgreSQL

Un cluster de bases de données postgreSQL contient une ou plusieurs bases nommées.

Seuls les utilisateurs et groupes d'utilisateurs sont partagés sur l'ensemble du cluster, aucune autre donnée n'est partagée.

Une connexion cliente à un serveur de base de données ne peut accéder qu'aux données d'une seule base, celle indiquée dans la requête de connexion.

Schémas

Une BD contient un ou plusieurs schémas nommés qui, eux, contiennent entre autres des tables.

Le même nom d'objet peut être utilisé dans différents schémas sans conflit.

Les schémas ne sont pas séparés de manière rigide : un utilisateur peut accéder aux objets de n'importe quel schéma de la BD à laquelle il est connecté, sous réserve qu'il en ait le droit.

Schémas

Toute nouvelle BD contient un schéma public.

Par défaut, les tables créées sont placées dans un schéma nommé public :

Les instructions suivantes sont équivalentes :

```
CREATE TABLE produits ( ... );
CREATE TABLE public.produits ( ... );
```

Schémas

Utilité de définir plusieurs schémas:

- autoriser plusieurs utilisateurs à utiliser une BD sans interférer les uns avec les autres ;
- organiser les objets de la BD en groupes logiques afin de faciliter leur gestion ;
- les applications tierces peuvent être placées dans des schémas séparés pour éviter les collisions avec les noms d'autres objets.

Objets dans un schéma

- Quand un objet est créé, il se voit affecter un propriétaire qui est normalement le rôle qui a exécuté la requête de création.
- Pour la plupart des objets, l'état initial est que seul le propriétaire (et les superutilisateurs) peut faire quelque chose avec cet objet.
- Pour permettre aux autres rôles de l'utiliser, des droits doivent être donnés.

Droits sur les objets d'une BD

SELECT
INSERT
UPDATE
DELETE
TRUNCATE
REFERENCES
TRIGGER
CREATE
CONNECT
TEMPORARY
EXECUTE
USAGE

Droits sur les objets d'une BD

- Le droit de modifier ou de détruire un objet est le privilège du seul propriétaire.
- Un objet peut se voir affecter un nouveau propriétaire avec la commande ALTER
  e.g. ALTER TABLE *tablename* OWNER TO *username*
- Les superutilisateurs peuvent toujours le faire. Les rôles ordinaires peuvent le faire seulement s'ils sont le propriétaire actuel de l'objet ou un membre du rôle propriétaire.

Gestion des droits d'accès

```
INSERT INTO Cinema.Studio(name)
        SELECT DISTINCT studioname
        FROM Kine.Movie
        WHERE studioname NOT IN (
                SELECT name FROM Kine.Studio );
```

Cinema et Kine sont des schémas,
La table Studio appartient au schéma Cinema,
les tables Movie et Studio au schéma Kine.

Gestion des droits d'accès

L'exécution de cette transaction nécessite les privilèges suivants :

- USAGE pour les schémas Cinema et Kine
- INSERT pour la table Studio (il serait suffisant de bénéficier de ce privilège pour l'attribut name de la table Studio)
- SELECT pour les tables Studio et Movie

Gestion des droits d'accès

- Le créateur d'un schéma possède tous les privilèges pour ce schéma
- L'utilisateur est identifié lors de sa connexion au serveur
- Des privilèges pourront être accordés à un module (programme d'application, e.g., session interactive, code SQL incorporé dans un texte du langage hôte, fonction ou procédure stockée) ou à un ensemble d'utilisateurs

Accord de droits : GRANT

GRANT *liste de privilèges*
ON liste d'*objets de la base*
TO *liste d'utilisateurs* ou PUBLIC
[WITH GRANT OPTION]

GRANT SELECT,INSERT ON marque TO durand WITH GRANT OPTION

L'utilisateur qui exécute une instruction GRANT doit posséder tous les privilèges accordés ainsi que la "GRANT OPTION" sur les objets concernés.

Révocation de droits : REVOKE

REVOKE [ALL] <privilèges list>
ON <database_name>
FROM <user_name>
[CASCADE | RESTRICT]

Accord & révocation de droits : exemple

User U: GRANT INSERT ON R TO V
User U: GRANT INSERT(A) ON R TO V
User U: REVOKE INSERT ON R FROM V

V conserve la possibilité d'insérer A dans R

Le retrait d'un privilège général n'ôte pas un privilège particulier

SQL DCL in a nutshell

CREATE ROLE *nom_utilisateur_ou_groupe* [INHERIT|NOINHERIT];
CREATE ROLE *nom_utilisateur_ou_groupe* SUPERUSER;
CREATE ROLE *nom_utilisateur_ou_groupe* LOGIN;
CREATE USER *nom_utilisateur*;
CREATE ROLE *nom_utilisateur_ou_groupe* CREATEDB;
GRANT *nom_groupe* TO *role1*, ... ;
REVOKE *nom_groupe* FROM *role1*, ... ;
SET ROLE *role*;
SET ROLE NONE;
RESET ROLE;
GRANT *droit* ON *objet* TO *role* [WITH GRANT OPTION];
REVOQUE *droit* ON *objet* TO *role*;

Transaction Control Language

GESTION DES TRANSACTIONS

Gestion de la concurrence

- Plusieurs utilisateurs sont autorisés à manipuler la même base de données
- Il faut s'assurer que les actions des uns ne seront pas préjudiciables aux autres
- Le contrôle de concurrence doit rendre le partage des données complètement transparent aux utilisateurs

Gestion de la concurrence

- Un SGBD peut fonctionner sur plusieurs machines … une machine peut tomber en panne.
- Le SGBD doit garantir que la base restera dans un état cohérent même après une panne : sûreté de fonctionnement et mécanisme de reprise (sur panne).

Transaction

- Dans une transaction, un ensemble d'opérations de lecture/écriture, opérées par un même client sur une base, sera considéré comme effectué en un seul instant (globalement) par rapport aux autres clients.
- L'exécution d'une transaction assure le passage de la base d'un état cohérent vers un état cohérent

Propriétés ACID

Garantissent qu'une transaction est exécutée de façon fiable

- Atomicité
- Cohérence
- Isolation
- Durabilité

Propriétés ACID

Atomicité

On n'exécute jamais partiellement une transaction :

soit toutes les opérations sont validées et toutes les mises à jour sont enregistrées dans la base,

soit aucune mise à jour n'est enregistrée dans la base.

Propriétés ACID

Cohérence

Les transactions préservent la cohérence, on passe d'un état consistant (toutes les contraintes sont respectées) à un autre état consistant

(les étapes intermédiaires internes à une transaction ne respectent pas forcement cette cohérence)

Propriétés ACID

Isolation

Une transaction s'exécute comme si elle était seule à accéder à la base; elle ne voit pas les opérations effectuées par les autres transactions en cours.

Les états intermédiaires d'une transaction ne peuvent pas être vues par les autres transactions.

Propriétés ACID

Durabilité

Les effets d'une transaction validée perdurent, quelques soient les problèmes logiciels ou matériels, et notamment après la fin de la transaction.

Propriétés ACID

- Une transaction assemble plusieurs étapes en une seule opération tout-ou-rien.
- Les états intermédiaires entre les étapes ne sont pas visibles par les transactions concurrentes.
- De plus, si un échec survient qui empêche le succès de la transaction, alors aucune des étapes n'affecte la base de données.

Contraintes d'Intégrité

Invariants pour les transactions

- Statiques : NOT NULL, PRIMARY KEY, UNIQUE, CHECK, DEFAULT, FOREIGN KEY, ON DELETE CASCADE, ON DELETE RESTRICT, ON DELETE SET NULL, ON UPDATE CASCADE
- Dynamiques : programmées (associées à triggers)

Transactions en SQL

BEGIN;

Initie un bloc de transaction

Toutes les instructions apparaissant après la commande BEGIN sont exécutées dans une seule transaction jusqu'à ce qu'une commande COMMIT (on exécute) ou ROLLBACK (on annule tout) explicite soit exécutée.

Transactions en postgreSQL

Par défaut (sans BEGIN), PostgreSQL exécute les transactions en mode autocommit :
chaque instruction est exécutée dans sa propre transaction et une validation (COMMIT) est traitée implicitement à la fin de l'instruction (si l'exécution a réussi, sinon une annulation est exécutée).

Si le mode autocommit est désactivé toute requête hors transaction démarre une transaction (BEGIN implicite) et elle n'est terminée que par un COMMIT ou un ROLLBACK (explicites)

Concurrence

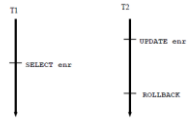
L'accès concurrent par plusieurs sessions à un objet partagé doit être contrôlé, pour éviter des problèmes bien connus:

- Lecture inconsistante ou lecture sale (dirty read) : la lecture des données est fausse
- Lecture non répétitive (non repeatable read) : deux lectures successives de la même donnée donnent des résultats différents
- Lecture de lignes fantômes (phantom) : une lecture retourne une ligne qui n'existe pas

Concurrence

Dirty read

Une transaction lit des données écrites par une transaction concurrente non validée



Concurrence

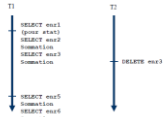
Non repeatable read

Une transaction relit des données qu'elle a lues précédemment et trouve que les données ont été modifiées par une autre transaction (validée depuis la lecture initiale)

Concurrence

Phantom read

Une transaction ré-exécute une requête renvoyant un ensemble de lignes satisfaisant une condition de recherche et trouve que l'ensemble des lignes satisfaisant la condition a changé du fait d'une autre transaction récemment validée.



Isolation

La norme SQL définit 4 niveaux d'isolation

Read Committed est le niveau d'isolation par défaut dans PostgreSQL

Niveau d'isolation	Lecture sale	Lecture non reproductible	Lecture fantome
READ_UNCOMMITTED	autorisée	autorisée	autorisée
READ_COMMITTED	interdite	autorisée	autorisée
REPEATABLE_READ	interdite	interdite	autorisée
SERIALIZABLE	interdite	interdite	interdite

Verrous

L'isolation est effectuée par verrouillage

- Chaque transaction demande des verrous de différents types sur les ressources dont elle dépend, telles que les lignes, les pages ou les tables. Les verrous demandés empêchent les autres transactions d'apporter aux ressources des modifications susceptibles de nuire à la transaction.
- Chaque transaction libère ses verrous lorsqu'elle ne dépend plus des ressources verrouillées.

Verrous

Un verrou est un objet à deux états (ouvert, fermé) supportant deux opérations atomiques :

- verrouiller(verrou) opération bloquante si le verrou est fermé (LOCK)
- déverrouiller(verrou) (UNLOCK)

Un verrou permet de contrôler l'accès à un objet en exclusion mutuelle : à un instant donné, une transaction au plus peut avoir accès à l'objet

Problèmes : famine, interblocage/étreinte fatale (deadlock)

Verrous

Granularités du verrouillage par une transaction: base, table, tuple, attribut, page disque

- Granularité fine (attributs, tuples) :
  un parallélisme élevé
  un surcoût en verrouillage et en place élevé
- Une grosse granularité (relations, base) :
  un parallélisme plus faible
  un surcoût négligeable

Verrous

Selon la norme SQL les verrous sont automatiquement réalisés par le système, le programmeur ne doit pas bloquer et débloquer lui-même les enregistrements.

Verrous

Typologie des verrous selon la norme SQL :

- Verrou partagé, shared lock
  utilisé pour lire 1 ou n lignes
- Verrou exclusif, exclusive lock
  utilisé pour modifier 1 ou n lignes
- Verrou global, global lock
  utilisé pour bloquer toutes les lignes d'une table

+ beaucoup de verrous intermédiaires hors norme, e.g. 12 type de verrous dans postgreSQL

Verrous

- Chaque requête essaye de poser des verrous sur les ressources dont elle a besoin et ne pourra s'exécuter que si elle a pu tous les acquérir.
- Le type de verrou dépend du type de requête.

Verrous

Verrou partagé: ACCESS SHARE

- Les commandes SELECT acquièrent un verrou partagé sur les tables référencées. En général, toute requête lisant seulement une table et ne la modifiant pas obtient ce mode de verrou.
- Il permet un autre accès en verrou partagé sur la même ligne, mais pas d'acquérir un autre type de verrou, sauf peut-être par celui qui l'a placé si nul autre n'en a placé un.

Verrous

Verrou exclusif: ROW EXCLUSIVE

- Il est utilisé pour modifier une ligne et est placé par les commandes INSERT, UPDATE et DELETE
- Une fois le verrou posé, il interdit tout autre accès à la ligne, pas même en lecture.

Verrous

Verrou global

- Il est utilisé pour modifier la structure d'une table par les instructions CREATE, ALTER et DROP
- Il bloque la table entière
- Aucune autre transaction ne peut plus accéder à rien dans la table.

Verrous

Des choix différents pour les différents SGBD :

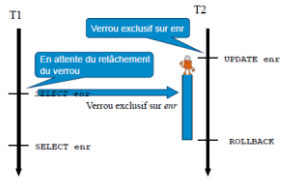
Beaucoup de différences entre postgreSQL, mySQL, Oracle si on y regarde de près.

Relâchement des verrous

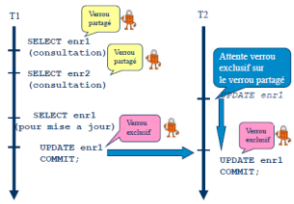
- Ce sont les instructions COMMIT et ROLLBACK qui relâchent les verrous (donc en fin de transaction)
- L'ensemble des verrous posés par les instructions d'une transaction sont relâchés à la fin de la transaction

  Utilisation à bon escient peut nécessiter de fractionner
- Pas de UNLOCK

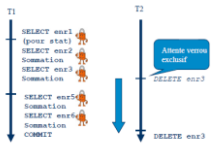
Solution pour la lecture inconsistante



Solution de la lecture non répétitive



Solution pour éviter les lignes fantômes

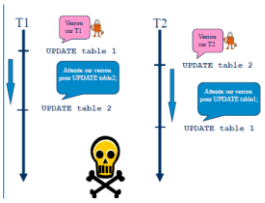


Verrous

Problèmes

- Le verrouillage de plusieurs ressources peut être bloquant et poser des problèmes d'étreintes fatales (deadlock)
- La sérialisation des transactions provoque une attente pour la libération de la ressource
  - dégradation de performance globale du système
  - risque de famine (livelock) : impossibilité de poser un verrou exclusif

Etreinte fatale (deadlock)



Traitement d'un deadlock

- Le deadlock doit être détecté par le SGBD (pb de détection de circuits)
- Le SGBD doit alors avertir l'utilisateur de la non possibilité de terminer la transaction
- Selon le SGBD
  - Annulation d'une transaction
  - Annulation des deux transactions
  - Message aux utilisateurs

Validité de transactions séquentielles

Théorème :

Principe de validité de transactions séquentielles

Si dans un univers concurrentiel, des transactions individuellement correctes sont exécutées séquentiellement, alors, à tout instant (entre deux transactions), la base sera cohérente vis-à-vis des contraintes.

Corollaire:

Un ordonnancement sérialisable est correct.

Visibilité des modifications en SQL

Visibilité externe

L'ensemble des modifications effectuées dans une transaction n'est rendu visible aux autres transactions (et rendu effectif dans la base) qu'au moment du commit qui termine la transaction.

L'ensemble des modifications effectuées dans une transaction est exécuté en un seul bloc, et sans recouvrement par d'autres commit (deux commit qui ont en commun la modification d'une même ligne seront exécutés séquentiellement).

Visibilité interne

Les modifications effectuées lors d'une transaction sont cependant visibles à l'intérieur de cette même transaction : toutes les modifications effectuées sont stockées dans une mémoire locale à la transaction, et tout se passe comme si la transaction travaillait sur une copie de la base.

SQL TCL in a nutshell

```
BEGIN;
COMMIT;
ROLLBACK;
```

BASE

Autre modèle de consistance, hors du monde relationnel, pour des systèmes distribués qui doivent être très disponibles

- (B)asically (A)vailable
- (S)oft state
- (E)ventually consistent

BASE

Basically Available

Quelle que soit la charge de la base de données (données/requêtes), le système garantit un taux de disponibilité de la donnée.

Des écritures peuvent être perdues, des lectures peuvent ne pas correspondre à la dernière version en date.

BASE

Soft state

La base peut changer lors des mises à jour ou lors d'ajout/suppression de serveurs. La base n'a pas à être cohérente à tout instant.

Eventually consistent

À terme, la base atteindra un état cohérent.