Introduction à la SGBD

Base de données & SQL

COLLEGE DE PARIS SUPÉRIEUR DAKARTECH

Dr Pape Abdoulaye **BARRO**

Enseignant – Chercheur

Spécialiste en Télémétrie et Systèmes Intelligents

Plan

- Généralités
- Le modèle conceptuel
- Le modèle relationnel
- Le langage SQL
- Etude et réalisation d’une base de données
- La sécurité des données

La sécurité des données

Objectif

- Dans cette section, nous allons traiter la sécurité des données qui est une notion fondamentale des bases de données.
- En effet, des pertes ou des modifications de données peuvent accidentellement ou intentionnellement causer de sérieux problèmes aux structures ou organisme dont les activités reposent essentiellement sur les données.
- Il convient alors de les protéger et aussi d'en assurer la disponibilité permanente.
  - Il s'agira de la protection de l'accès à la machine d'un point de vue physique ou réseau, les aspects de durée de vie du système ainsi que la politique de sauvegarde ;
  - De la politique de restriction d'accès aux données du SGBD par des comptes et l'utilisation des vues ;
  - De la capacité des outils du SGBD à protéger les opérations sur les données, à les restaurer et à revenir en arrière, comme pour les transactions.
  - Il convient également d'aborder la notion de trigger pour assurer une bonne gestion du contenu des données lors des opérations d'insertion, de mise à jour ou de suppression de données.

La sécurité des données

Contrôle d'accès et sauvegarde

On décrit dans cette partie, plusieurs types de préoccupations:

- **L'accès physique à la machine**

Un accès physique permet généralement de contourner toutes les protections liées au système d'exploitation ou au SGBD en question.

- Il suffit de démarrer une machine avec un autre système d'exploitation présent sur un CD ou une clé USB par exemple pour obtenir un accès au disque et ainsi aux données associées au SGBD.
  - La première précaution à prévoir consiste à protéger le BIOS (Basic Input Output System) ou l'équivalent de la machine pour l'empêcher de démarrer sur un autre disque que celui qui contient le « bon » système d'exploitation.
- Le fait de pouvoir accéder directement à la machine permet également à une personne animée de mauvaises intentions d'empêcher l'accès aux données en détruisant physiquement le disque sur lequel se trouvaient les fichiers ou même la machine.
  - Par conséquent, la ou les machines qui contiennent le SGBD et les fichiers de données doivent se trouver dans des pièces dont l'accès est, au minimum, contrôlé.
- L'alimentation électrique ne doit pas aussi être accessible.

Ces précautions ne sont pas toujours d'ordre purement technique; elles relèvent souvent du simple bon sens commun.

La sécurité des données

Contrôle d'accès et sauvegarde

- **L'accès distant à la machine à travers le réseau**

L'accès à la machine par le réseau est devenu une porte d'entrée privilégiée pour les personnes malintentionnées, car rares sont les serveurs qui ne sont pas connectés aujourd'hui. On distingue ainsi principalement deux aspects de problèmes de sécurité :

- les attaques « internes » qui proviennent de personnes ayant accès à la machine par un compte standard associé à un mot de passe correct ;
- les attaques « externes » qui se font en profitant d'une faille du système d'exploitation ou d'une des applications installées sur la machine.

Dans ces cas de figure, les protections dépendent spécifiquement de l'administrateur système et réseau et pas forcement de l'administrateur du SGBD.

- Le système et les applications présentes sur la machine doivent être maintenus et mis à jour régulièrement pour éviter les problèmes.
- L'administrateur système doit également tenir à jour les comptes utilisateurs qui possèdent les droits d'accès à la machine et vérifier la qualité des mots de passe associés.
  - Classiquement, une intrusion se fait en utilisant un compte « oublié », créé par exemple pour un besoin ponctuel, dont le mot de passe est assez simple pour être deviné.

La sécurité des données

Contrôle d'accès et sauvegarde

- **La pérennité du système**

Une base de données se conçoit généralement pour de nombreuses années d'utilisation. Il est donc essentiel, avant de choisir une machine, son système d'exploitation ainsi que le logiciel de SGBD, de prendre en considération leurs pérennités respectives.

Le choix peut alors se porter sur des ensembles moins performants mais que l'on considère comme plus viables en termes de durée de vie. Parmi lesquels, nous considérons:

- Le suivi de la machine elle-même: la durée de la garantie matérielle, la possibilité de retrouver les pièces etc.
- La durée de vie du système d'exploitation: Un système qui n'est plus mis à jour par son éditeur et qui présente des trous importants de sécurité se révèle un très mauvais choix.
- L'évolutivité du SGBD: Un SGBD propriétaire qui n'offre pas les échanges de données avec d'autres logiciels handicape la migration vers un autre SGBD. De même que pour les points précédents, un SGBD qui présente des failles importantes de sécurité d'accès ou de fonctionnement et qui n'est plus mis à jour par l'éditeur constitue un problème potentiel qu'aucune autre précaution ne saurait combler.

La sécurité des données

Contrôle d'accès et sauvegarde

- **Sauvegarde des données**

Des copies de sauvegarde des données contenues dans la base, mais aussi des métadonnées (dictionnaire de données) du SGBD, doivent être réalisées régulièrement.

  - Outre les problèmes de malveillance, un incendie ou une inondation peuvent réduire à néant des années de travail.
    - Il faut disposer d'une armoire de stockage adaptée à l'abri des inondations. Si ce stockage se trouve dans un lieu différent, c'est encore mieux.

- **Redondance des données**

Un procédé plus efficace et plus sûr consiste à disposer de copies de la base en plusieurs endroits géographiquement distincts. La plupart des SGBD sont capables d'effectuer des mises à jour sur des copies de la base de données situées sur d'autres machines via le réseau.

Cette méthode permet de plus d'améliorer la disponibilité des données :

  - Si l'un des serveurs est inaccessible, un autre peut prendre le relais.
  - Il est possible ainsi de répartir la charge sur plusieurs serveurs.

La sécurité des données

Limitations d'accès au SGBD

Les limitations concernent les autorisations accordées ou non sur les différents éléments que contient le SGBD et vont permettre de contrôler l'accès général au SGBD.

- **La gestion des utilisateurs**

Les instructions qui permettent d'affecter les droits sont définies dans la norme SQL.

- Classiquement, pour protéger une ressource, on commence par en interdire l'accès à tous. Puis on décrit ceux qui sont autorisés à l'utiliser (utilisateurs). On associe ensuite à chaque utilisateur un identifiant (mot de passe) pour éviter les emprunts d'identité.
- Le SGBD stocke les informations suivantes pour chaque objet qu'il contient :
  - Le type de droit: Sélection, création, destruction…
  - L'objet sur lequel s'appliquent les droits: Une base de données, une table, une vue…
  - Le nom de l'utilisateur.
  - L'identifiant, au sens du mot de passe.
  - Le donneur des droits.
- Pour créer un utilisateur avec la norme SQL, c'est sous la forme suivante:
  - CREATE USER <type de droit> IDENTIFIED BY <identifiant> ;
  - DROP USER permet de détruire l'utilisateur créé.
- Un utilisateur peut prendre le(s) rôle(s) dont il a besoin (rôle) pour pouvoir travailler sur les objets du SGBD qui le concerne. Pour créer un rôle, on fait:
  - CREATE ROLE consultation_seulement ;

La sécurité des données

Limitations d'accès au SGBD

- **La gestion des droits**

Une fois que l'utilisateur est connecté au système, il convient de gérer son accès aux données. Les droits sont distribués à des utilisateurs, des groupes ou des rôles.

- Avec SQL, il est possible de spécifier des droits essentiellement sur les opérations de manipulation de tables (ou de vues considérées comme des tables) suivantes :
  - interrogation (SELECT) ;
  - insertion (INSERT) ;
  - mise à jour (UPDATE) ;
  - destruction (DELETE) ;
  - référence à une table (REFERENCE).
- L'instruction GRANT permet de réaliser l'association des droits et du bénéficiaire. Sa forme générale est la suivante :
  - GRANT <type de droit> ON <objet> TO <nom utilisateur>

La sécurité des données

Limitations d'accès au SGBD

Créons un utilisateur nommé my_user:

- CREATE USER my_user IDENTIFIED BY password;
- L'option 'ALL PRIVILEGES' sert à donné tous les droits à l'utilisateur. L'administrateur dispose de tous les droits sur tous les objets. Exemple:
  - CREATE DATABASE ma_base;
  - GRANT ALL PRIVILEGES ON ma_base.* TO 'my_user' ;
  - CREATE USER user_2 IDENTIFIED BY password_2;
  - USE ma_base;
  - CREATE TABLE ma_classe (NumClasse **INT PRIMARY KEY**, Titre **CHAR**(50)) ;
  - GRANT SELECT ON ma_classe TO 'user_2' ; (ERREUR)
- L'utilisateur my_user n'a pas le droit de retransmettre ses droits à un autre utilisateur. L'administrateur aurait dû entrer la commande suivante :
  - GRANT ALL PRIVILEGES ON ma_base.* TO 'my_user WITH GRANT OPTION ;
- Il pourra alors attribuer des droits aux autres utilisateurs:
  - GRANT SELECT ON ma_classe TO 'user_2' ; (CORRECTE)
- On peut donner un droit à **tous les utilisateurs du SGBD** avec le mot clé PUBLIC, comme suit:
  - GRANT SELECT ON ma_classe TO PUBLIC;
- On peut affiner ces permissions en ne les accordant que pour certains champs de la table.
  - GRANT UPDATE Titre ON ma_classe TO 'user_2' ;

La sécurité des données

Limitations d'accès au SGBD

L'instruction GRANT permet également de donner un rôle à un utilisateur ou à un ensemble d'utilisateurs et s'utilise alors de la manière suivante :

- GRANT <rôle> TO <liste des noms d'utilisateur séparés par des virgules>
- On crée les rôles et on met à jour les droits nécessaires. On peut considérer les rôles suivant (**consultation** : interrogation ; **utilisation** : mise à jour et insertion ; et **gestion** : destruction) sur ma_classe.
  - CREATE ROLE consultation;
  - CREATE ROLE utilisation;
  - CREATE ROLE gestion;
  - GRANT SELECT ON ma_classe TO consultation;
  - GRANT UPDATE, INSERT ON ma_classe TO utilisation;
  - GRANT DELETE ON ma_classe TO gestion;
- On affecte les rôles aux utilisateurs.
  - GRANT consultation TO 'my_user', 'user_2' ;
  - GRANT utilisation TO 'my_user', 'user_2' ;
  - GRANT gestion TO 'my_user';
- Pour retirer les droits accordés par l'instruction GRANT, on utilise l'instruction REVOKE. Elle est de la forme suivante :
  - REVOKE <type de droit> ON <objet> FROM <nom utilisateur>;
  - **Exemple**: REVOKE SELECT ON ma_classe FROM 'user_2' ;
- On peut affiner ces permissions en ne les accordant que pour certains champs de la table.
  - GRANT UPDATE Titre ON ma_classe TO 'user_2' ;

La sécurité des données

Limitations d'accès au SGBD

- **Utilisation des vues**

Une approche complémentaire consiste à définir plus finement les objets sur lesquels on affecte les droits en se servant simplement des vues SQL. Celles-ci permettent de « cacher » certaines données à l'aide de critères de sélection précis. De plus, les vues permettent de masquer aux utilisateurs la complexité de la structure des données, leur fournissant ainsi une interface plus commode avec la base de données. Nous allons manipuler la base de données 'casse', en guise d'exemple.

- **Vue utilisant une jointure simple**

Le service marketing veut vérifier s'il n'y a pas de corrélations entre la couleur des véhicules vendus et la ville dans laquelle résident les clients qui les achètent. Il est inutile de fournir à ce service la procédure pour obtenir ces informations qui nécessite de lier les trois tables 'voiture', 'vente' et 'client'.

- On crée une vue qui contient uniquement la projection de ces deux informations et qui sélectionne les voitures vendues.
  - CREATE VIEW couleur_ville (Couleur, Ville) AS SELECT voiture.couleur, client.ville FROM voiture JOIN vente JOIN client ON voiture.NumVoit=vente.Numvoit AND client.NumAch=vente.Numach ;
- On donne les droits de visualiser ces informations aux personnes du service marketing.
  - GRANT SELECT ON couleur_ville TO 'my_user', 'user_2' ;

La sécurité des données

Limitations d'accès au SGBD

- **Vue utilisant une jointure plus élaborée**

On souhaite que tous les utilisateurs puissent consulter le catalogue des voitures non encore vendues. Cette liste de voitures est accessible par une requête plus complexe que la précédente, de type jointure externe. Il s'agit d'un cas d'utilisation d'une vue, très pratique pour les utilisateurs qui n'ont pas à connaître les notions avancées de jointure externe.

  - CREATE VIEW catalogue (NumVoit, Marque, Type, Couleur) AS SELECT voiture.NumVoit, voiture.Marque, voiture.Type, voiture.Couleur FROM voiture LEFT OUTER JOIN vente ON voiture.NumVoit=vente.Numvoit WHERE vente.Numvoit IS NULL;
- On donne la permission à l'utilisateur PUBLIC, c'est-à-dire à tous les utilisateurs.
  - GRANT SELECT ON catalogue TO PUBLIC;

La sécurité des données

Limitations d'accès au SGBD

- **Vue avec possibilité de mise à jour**

Le service comptabilité doit pouvoir accéder aux informations de vente et de clientèle afin de facturer et calculer le chiffre d'affaires. En outre, ce service a besoin de mettre à jour le prix de vente.

- L'option CHECK OPTION permet les mises à jour des données au travers de la vue.
  - CREATE VIEW journal_compta (Date_de_vente, Prix_de_vente, Nom_client, Ville_client) AS SELECT vente.DateVent, vente.Prix, client.Nom, client.Ville FROM vente JOIN client ON vente.NumAch=client.NumAch WITH CHECK OPTION ;
- On donne les droits à un rôle que les utilisateurs vont utiliser.
  - CREATE ROLE comptabilite ;
  - GRANT SELECT ON journal_compta TO comptabilite;
  - GRANT UPDATE Prix_de_vente ON journal_compta TO comptabilite;
- On donne le rôle 'comptabilité' aux utilisateurs
  - GRANT comptabilite TO 'my_user', 'user_2';

La sécurité des données

Transactions

Nous présentons dans cette section, plusieurs outils procurés par les SGBD **pour protéger les opérations** effectuées sur les données. Les incidents liés aux données dans un SGBD proviennent essentiellement de l'accès concurrent à ces dernières par plusieurs utilisateurs. Ces problèmes sont habituellement résolus par les mécanismes associés aux transactions. Elles permettent également de résoudre les pertes dues aux erreurs de manipulation ou celles liées aux erreurs dans le traitement des données, par exemple en cas de panne matérielle.

- **Accès concurrentiel aux données**

  En définissant des accès et des permissions sur les différents objets gérés par le SGBD, on a donné accès à plusieurs utilisateurs ou applications, en même temps, aux données. De plus, la plupart des machines sont connectées à un réseau, ce qui augmente encore le nombre potentiel d'utilisateurs simultanés.

  Nous allons voir quelques types d'incohérences qui peuvent être provoquées par l'accès concurrent aux données.

La sécurité des données

Transactions

- **Lecture étrange**

L'accès multiple en lecture ne pose pas habituellement de problèmes, mais que se passe-t-il lorsqu'un ou plusieurs utilisateurs décident de modifier les mêmes données au même moment ?

On considère la séquence d'instructions suivante appliquée à la base de données « **casse** ». On suppose que tous les utilisateurs disposent de tous les droits sur tous les objets (tables et champs) de cette base de données :

- L'utilisateur 'my_user' consulte la liste des voitures non vendues.
- L'utilisateur 'user_2' enregistre la vente d'une voiture.
- L'utilisateur 'my_user' relance la même requête et trouve un résultat différent.
- L'utilisateur 'user_2' invalide la vente de cette voiture.
- L'utilisateur 'my_user' pensant s'être trompé relance la même requête qui aboutit au résultat initial.

Pour trois exécutions de la même requête, l'utilisateur 'my_user' va obtenir des résultats différents.

On pourrait utiliser ici le terme de lecture fantôme pour qualifier les problèmes rencontrés : **une donnée apparaît puis disparaît**.

La sécurité des données

Transactions

- **Incohérence de résultats**

On peut imaginer une autre série d'instructions qui réalisent des modifications de données effectuées par différents utilisateurs. En raison de l'augmentation des frais de structure, le service comptable a décidé d'une augmentation générale du prix de vente de 5%.

Afin que cette dernière passe inaperçue et dans le cadre d'une campagne de communication, le service marketing offre 100 euros de ristourne sur tout le catalogue pendant un mois.

Supposons que l'utilisateur 'my_user' appartient au service comptable et l'utilisateur 'user_2' au service marketing. Les vérifications sont effectuées par un troisième utilisateur 'user_3' de la direction.

- L'utilisateur 'my_user' consulte le prix de vente de la voiture 'X'.
- L'utilisateur 'user_2' effectue la modification de promotion sans vérifier le prix de vente (Prix_vente = Prix_vente – 100).
- L'utilisateur 'my_user' effectue la modification d'augmentation (Prix_vente = Prix_vente × 1,05).
- L'utilisateur 'user_3' vérifie le résultat de l'opération de modification du prix de vente et constate une différence avec le résultat attendu: le prix est égal à (Prix_vente – 100) × 1,05 alors qu'il s'attendait à un prix égal à (Prix_vente × 1,05) – 100.

On obtient alors une « incohérence » due à la séquence de modification des données.

La sécurité des données

Transactions

- **Verrous**

On a donc besoin de disposer d'un mécanisme qui garantisse une certaine « **exclusivité** » sur les données lors des opérations de mise à jour: **il s'agit des verrous**.

L'idée est simple: **on bloque une ressource pour effectuer les opérations et on la libère dès que les opérations sont effectuées**.

Même si cette méthode semble résoudre le problème; elle présente cependant quelques pièges et doit être utilisée avec prudence. En effet, on peut rapidement parvenir à une situation de blocage que l'on nomme **étreinte fatale** (deadlock en anglais) ou parfois **interblocage**. Pour illustrer cette situation, on suppose que deux vendeurs veulent mettre à jour la base de données '**casse**' :

- L'utilisateur 'my_user' veut insérer une vente de voiture dans la table 'vente' et constate à cette occasion une erreur sur la couleur dans la table 'voiture' qu'il veut modifier.
- L'utilisateur 'user_3' veut insérer à la fois une nouvelle voiture dans la table 'voiture' et la mention de sa vente dans la table 'vente'.

La séquence d'instruction peut être la suivante :

- L'utilisateur 'my_user' pose un verrou sur la table 'vente'.
- L'utilisateur 'user_3' pose un verrou sur la table 'voiture'.
- L'utilisateur 'my_user' a terminé la mise à jour de 'vente' et veut réaliser celle de 'voiture'… mais la table 'voiture' est verrouillée par 'user_3'.
- L'utilisateur 'user_3' a terminé la mise à jour de 'voiture' et veut procéder à celle de 'vente'… mais la table 'vente' est verrouillée par 'my_user.

On parvient à une situation où les deux utilisateurs attendent que l'autre libère la ressource pour continuer.

La sécurité des données

Transactions

D'autres phénomènes de blocage peuvent survenir dans l'utilisation des verrous. On parle parfois dans ce cas de **famine**. Voici un exemple illustrant cette possibilité (algorithme des « lecteurs-rédacteurs »).

- Tous les lecteurs peuvent lire en même temps.
- Si un rédacteur veut écrire, aucun lecteur ne doit plus utiliser la donnée.
- Si un rédacteur est en train d'écrire, aucun lecteur ne peut lire la donnée et aucun autre rédacteur ne peut y accéder (accès exclusif).

S'il existe un grand nombre de lecteurs qui se succèdent en lecture sur la donnée, l'attente pour un rédacteur qui voudrait la modifier peut alors devenir quasi infinie.

Laisser la gestion des verrous à un utilisateur est donc délicat et peut conduire à des blocages du système. Les SGBD performants disposent d'outils capables de détecter et de résoudre ces phénomènes, voire de les prévenir le cas échéant.

La sécurité des données

Transactions

- **Mécanisme des transactions**

Nous allons essayer de résoudre les problèmes d'accès concurrents vus précédemment, en utilisant les transactions. L'idée est de considérer un ensemble d'instructions comme une seule instruction. L'ensemble d'instructions est dit unitaire ou atomique.

- **Propriétés des transactions**

La notion d'atomicité peut être décrite en ces termes :

- Soit le SGBD est capable d'exécuter toutes les instructions qui composent une transaction et il effectue les mises à jour provoquées par ces instructions.
- Soit il n'y parvient pas et il remet la base de données dans l'état cohérent précédent le début de l'exécution des instructions de la transaction.

Les propriétés que doivent vérifier les transactions sont résumées par le terme ACID :

- **Atomicité**. Une transaction est atomique : elle est exécutée entièrement ou abandonnée.
- **Cohérence**. La transaction doit se faire d'un état cohérent de la base vers un autre état cohérent.
- **Isolement**. Des transactions simultanées ne doivent pas interférer entre elles.
- **Durabilité**. La transaction a des effets permanents même en cas de panne.

La sécurité des données

Transactions

- **Réalisation des transactions dans les SGBD**

Pour mettre en œuvre les transactions, le SGBD doit offrir l'exclusivité d'accès aux données ainsi que la possibilité d'annuler des modifications effectuées. Afin de garantir l'accès exclusif aux données, le SGBD utilise les verrous vus précédemment. Ces mécanismes sont associés à des algorithmes de protection sophistiqués afin de prévenir les problèmes des méthodes de verrouillage.

La possibilité de revenir en arrière repose sur l'utilisation d'un journal de transactions. Pour ce faire, le SGBD garde une trace de toutes les requêtes de la transaction et des données qui sont modifiées par ces instructions. On conserve donc un journal de l'ensemble des lectures et écritures afin de pouvoir reconstituer un système cohérent. En utilisant ce journal, à partir d'un état courant du système, on est capable de réexécuter les instructions qui n'ont pu l'être effectivement ou on revient en arrière si ce n'est pas possible.

La sécurité des données

Transactions

- **Différents niveaux de transaction**

La norme SQL fixe plusieurs degrés de qualité pour les transactions: on parle de **niveaux d'isolation**. Les niveaux standard sont les suivants :

- 0, **READ UNCOMMITED**. Lecture de données non validées en cours de transaction (niveau le plus bas).
- 1, **READ COMMITED**. Modification des données possible en cours de transaction.
- 2, **REPEATABLE READ**. Insertion de nouveaux enregistrements possible en cours de transaction.
- 3, **SERIALIZABLE**. Toutes les transactions sont traitées en « série » (niveau le plus sûr).

La sécurité a un coût en matière de performances et, dans certains contextes, il n'est pas nécessaire de demander au SGBD de traiter une requête avec le niveau maximal d'exclusivité. Il est clair que, pour une simple opération de lecture, le niveau 1 qui garantit l'impossibilité de lire des données non validées peut être suffisant.

La sécurité des données

Transactions

- **Syntaxe SQL des transactions**

L'instruction pour débuter une transaction est **START TRANSACTION**. La transaction prendra fin par l'une des instructions suivantes :

- **COMMIT**. Les instructions effectuées depuis START TRANSACTION sont validées.
- **ROLLBACK**. Les instructions effectuées depuis START TRANSACTION sont annulées.

Voici un exemple d'insertion et de mise à jour dans la base de données '**casse**' que l'on annule ensuite :

- #Démarrage de la transaction
  - START TRANSACTION ;
- #Insertion d'un tuple dans la table 'client'
  - INSERT INTO client VALUES (6,'Laetitia',34,'Paris','F');
- #Mise à jour des données de prix (augmentation de 5 %)
  - UPDATE vente SET Prix=Prix*1.05 ;
- #Annulation des modifications faites depuis START TRANSACTION
  - ROLLBACK ;

Voici un exemple de destruction dans la table 'client' de la base de données '**casse**' que l'on valide ensuite :

- #Démarrage de la transaction
  - START TRANSACTION ;
- #Destruction des tuples dont le champ 'Ville' vaut 'Paris' dans la table 'client'
  - DELETE FROM client WHERE Ville='Paris';
- #Validation des modifications faites depuis START TRANSACTION
  - COMMIT ;

On peut choisir le niveau d'isolation au moment où l'on démarre la transaction. Dans l'exemple suivant, on fixe le niveau maximal de sécurité.

- START TRANSACTION
- ISOLATION LEVEL SERIALIZABLE ;
- …

La sécurité des données

Transactions

- **« Points de retour » dans les transactions**

Les instructions contenues dans une transaction sont considérées par le SGBD comme « unitaires », c'est-à-dire qu'elles sont exécutées d'un seul bloc ou ne sont pas exécutées. L'instruction ROLLBACK annule toutes les instructions effectuées depuis l'instruction START TRANSACTION. Il est possible de diviser cet ensemble d'instructions en sous ensembles par la définition de « **points de retour** » utilisés par l'instruction ROLLBACK. Au lieu d'annuler les instructions exécutées depuis START TRANSACTION, on revient à l'état de la base de données depuis le point défini par l'instruction **SAVEPOINT**.

- #Démarrage de la transaction
  - START TRANSACTION ;
- #Insertion d'un tuple dans la table 'client'
  - INSERT INTO client VALUES (8,'Annabelle',29,'Paris','F');
- # Définition d'un point de retour que l'on nomme 'UN'
  - SAVEPOINT 'UN'
- #Mise à jour des données de prix (augmentation de 5 %)
  - UPDATE vente SET Prix=Prix*1.05 ;
- #Destruction des tuples dont le champ 'Ville' vaut 'Paris' dans la table 'client'
  - DELETE FROM client WHERE Ville='Paris';
- #Annulation des modifications faites depuis le point 'UN'
  - ROLLBACK TO 'UN' ;
  - …

Une différence essentielle par rapport à l'instruction ROLLBACK vue précédemment est que, dans ce cas, la transaction n'est pas terminée. Un autre ROLLBACK reviendrait à l'état de la base de données avant l'exécution de l'instruction START TRANSACTION.

Le mécanisme des points de retour est très utile pour prévenir les cas de mauvaise manipulation de données. On combine ainsi la possibilité d'effectuer les modifications dans une seule transaction avec celle de ne pas annuler toutes les modifications. Certaines d'entre elles prennent en effet un temps considérable.

La sécurité des données

Triggers

Un trigger est un ensemble d'instructions SQL, définies par l'utilisateur, qui seront déclenchées lors d'une action d'ajout, de suppression ou de mise à jour de données. Il s'applique donc à un objet de type table. On peut choisir d'exécuter un trigger avant ou après une instruction de mise à jour.

Les données manipulées par le trigger sont stockées dans des tables temporaires que l'on désigne dans le code du trigger par les termes:

- **NEW** valeurs, après l'exécution de l'opération de mise à jour ;
- **OLD** valeurs, avant l'exécution de l'opération de mise à jour.

- **Syntaxe générale pour la création et la destruction d'un trigger**

L'instruction, normalisée par la norme SQL, est de la forme générale suivante :

- CREATE TRIGGER 'Nom du trigger'
- 'Moment où le trigger est exécuté'
- 'Opération concernée'
- ON 'Nom de la table'
- FOR EACH ROW(ou STATEMENT)
- BEGIN
- 'Instructions'
- END
  - Le moment peut prendre les valeurs BEFORE ou AFTER.
  - Les opérations concernées sont INSERT, UPDATE et DELETE.
  - La différence entre ROW et STATEMENT est que l'on exécute les instructions pour chaque ligne ou pour toute la table.

Pour détruire un trigger, on utilise l'instruction **DROP** et on spécifie le nom du trigger que l'on veut détruire.

- DROP TRIGGER conversion

La sécurité des données

Triggers

- **cas d'utilisation**
  - **Contrôle de la forme du contenu des champs**

Pour le formatage, on veut imposer que les noms d'une ville dans la table 'client' soient transformés en majuscules avant d'être insérés. On fait référence ici à la nouvelle table 'NEW' que l'on modifie avant l'insertion.

```
CREATE TRIGGER ville_majuscule BEFORE INSERT ON client FOR EACH ROW
BEGIN
SET NEW.Ville=UPPER(NEW.Ville) ;
END
```

  - **Modification automatique du contenu d'un champ**

Les triggers permettent, par exemple, de compléter le mécanisme d'intégrité référentielle par des contrôles plus fins et la définition des actions associées à ces derniers. On considère l'exemple de la base de données 'casse'. Si le prix de vente est supérieur à 40 000 lors d'une insertion de données dans la table 'vente', on suppose que la personne a saisi par erreur le prix en francs et non pas en euros. On effectue automatiquement la conversion des francs en euros. Là encore, on modifie les données de la table 'NEW' qui contient celles que l'on va insérer.

```
CREATE TRIGGER conversion BEFORE INSERT ON vente FOR EACH ROW
BEGIN
IF New.Prix> 40000
THEN SET
NEW.Prix=NEW.Prix/6.55 ;
END IF;
END
```

La sécurité des données

Triggers

- **Mise à jour d'autres tables**

On peut également provoquer la mise à jour d'autres tables à l'aide d'un trigger. Si l'on ajoute un champ 'compte' à la table 'client' qui représente l'état du solde de son compte vis- à-vis de l'entreprise, on veut pouvoir mettre à jour automatiquement le champ 'compte' de l'acheteur lors de l'insertion d'une opération de vente. À cette occasion, on soustrait le prix de vente de la voiture du montant du compte du client. C'est la seule manière de réaliser cette opération de modification d'une autre table lors d'une simple insertion.

Ajout d'un champ 'compte' à la table 'client'

- ALTER TABLE client ADD COLUMN compte INT ;

Création du trigger

```
CREATE TRIGGER compte BEFORE INSERT ON vente FOR EACH ROW
BEGIN
UPDATE client
SET client.Compte=client.Compte – NEW.Prix
WHERE client.NumAch=New.NumAch ;
END
```

Ajout d'une vente dans la table 'vente'

- INSERT INTO vente VALUES (6, '2005-03-01',50000,2,5)

- Le compte du client de numéro 'NumAch' égal à '2' sera automatiquement diminué de la somme de '50000'.

Du point de vue de la sécurité des données, le code associé au trigger s'exécute sur le serveur sans interaction avec le client. On réduit ainsi les risques d'erreur liés aux échanges entre le client et le serveur.

FIN

Feedback: pape.abdoutaye.barro@gmail.com

COLLEGE DE PARIS SUPÉRIEUR
DAKARTECH

Dr Pape Abdoulaye **BARRO**
Enseignant – Chercheur
Spécialiste en Télémétrie et Systèmes Intelligents