Secteur Tertiaire Informatique
Filière étude - développement

Activité « Développer la persistance des données »

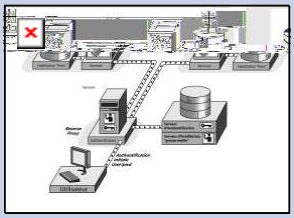
Mise en œuvre des transactions

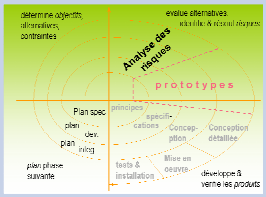
Accueil

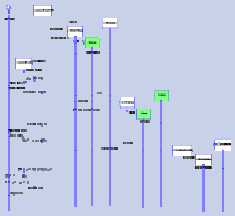
Apprentissage

Période en entreprise

Evaluation







Code barre

SOMMAIRE

I TRANSACTIONS ET VERROUS

Toutes les modifications de données dans SQL SERVER sont effectuées dans le cadre de transactions. Par défaut, SQL SERVER démarre une transaction pour chaque instruction individuelle et la valide automatiquement si l'exécution de l'instruction se termine normalement.

Une transaction est caractérisée les critères ACID

- **A**tomique : si une des instructions échoue, toute la transaction échoue
- **C**ohérente, car la base de données est dans un état cohérent avant et après la transaction, c'est-à-dire respectant les règles de structuration énoncées.
- **I**solée : Les données sont verrouillées : il n'est pas possible depuis une autre transaction de visualiser les données en cours de modification dans une transaction.
- **D**urable : les modifications apportées à la base de données par une transaction sont validées

Par exemple, une transaction bancaire peut créditer un compte et en débiter un autre, ces actions devant être validées ensemble

Les transactions utilisateurs sont implémentées sous TRANSACT SQL grâce aux instructions **BEGIN TRANSACTION**, **COMMIT TRANSACTION** et **ROLLBACK TRANSACTION.**

Les verrous empêchent les conflits de mise à jour :

- Ils permettent la sérialisation des transactions de façon à ce qu'une seule personne à la fois puisse modifier un élément de données ; dans un système d réservation de places, les verrous garantissent que chaque place n'est attribuée qu'à une seule personne.
- SQL Server définit et ajuste de manière dynamique le niveau de verrouillage approprié pendant une transaction : il est également possible de contrôler manuellement le mode d'utilisation de certains verrous.
- Les verrous sont nécessaires aux transactions concurrentes pour permettre aux utilisateurs d'accéder et de mettre à jour les données en même temps.

Le contrôle de la concurrence permet de s'assurer que les modifications apportées par un utilisateur ne vont pas à l'encontre de celles apportées par un autre.

- Le contrôle pessimiste verrouille les données qui sont lues en vue d'une mise à jour, empêchant tout autre utilisateur de modifier ces données
- Le contrôle optimiste ne verrouille pas les données : si les données ont été modifiées depuis la lecture, un message d'annulation est envoyé à l'utilisateur.

Le choix entre ces deux types de contrôle est effectué en fonction de l'importance du risque de conflit de données et de l'évaluation du coût de verrouillage des données (nombre d'utilisateurs, nombre de transactions, temps de réponse …)

Le type de contrôle sera explicité en spécifiant le niveau d'isolement de la transaction.

II LE JOURNAL DES TRANSACTIONS

Le journal des transactions enregistre les modifications de données au fur et à mesure qu'elles sont effectuées.

Lorsqu'une modification de données est envoyée par l'application,si les pages de données concernées n'ont pas été chargées en mémoire cache à partir du disque lors d'une précédente modification, elles sont alors amenées en mémoire.

Chaque instruction est enregistrée dans le journal et la modification est effectuée en mémoire.

Le processus de point de contrôle reporte périodiquement dans la base de données toutes les modifications effectuées

A un instant t, il existe dans le journal :

- des transactions validées avant le dernier point de contrôle et donc reportées dans la base
- des transactions validées après le dernier point de contrôle, et donc non reportées dans la base
- des transactions non validées

En cas de défaillance du système, le processus de restauration automatique utilise le journal des transactions pour reprendre toutes les transactions validées non reportées dans la base, et annuler les transactions non validées.

III SYNTAXE

BEGIN TRAN[SACTION] [nom_transaction] [WITH MARK]

Cette instruction marque le début de la transaction.
Le nom de transaction n'est pas obligatoire, mais il facilite la lecture du code.
L'option WITH MARK permet de placer le nom de la transaction dans le journal des transactions. Lors de la restauration d'une base de données à un état antérieur, la transaction marquée peut être utilisée à la place d'une date et d'une heure

COMMIT TRAN[SACTION] [nom_transaction]

Cette instruction valide la transaction : les modifications sont effectives dans la base de données.

ROLLBACK TRAN[SACTION] [nom_transaction]

Cette instruction annule la transaction : les données de la base sont celles d'avant les ordres de modification, insertion ou suppression.

SAVE TRAN[SACTION] [nom_point_de_sauvegarde]

Cette instruction permet de définir un point d'enregistrement à l'intérieur d'une transaction.

Exemple 1: définition d'une transaction assurant le transfert de fonds entre le compte chèque et le compte épargne d'un client.

```
BEGIN TRAN
        Exec debit(100, cptCourant)
        Exec credit(100, cptEpargne)
COMMIT TRAN
```

Les deux opérations débit et crédit ne seront validées qu'après le COMMIT TRAN.

Exemple 2: Définition d'une transaction lors de l'ajout d'une ligne de ventes et la mise à jour de la table articles pour la quantité vendue.

Structure des tables :

VENTES (vnt_art, vnt_cli, vnt_qte, vnt_prix)

ARTICLES (art_num, art_nom, art_coul, art_pa, art_pv, art_qte, art_frs)

BEGIN TRANSACTION

```
BEGIN TRY
-- Opération d'insertion dans ventes
   INSERT Ventes(Vnt_Clt,Vnt_mag,Vnt_Art,Vnt_Qte,Vnt_Prix,Vnt_Date)
   values(@pCodClt,@pCdMag,@pCodArt,@pQte,@pPrix,@pDate)
END TRY
BEGIN CATCH
   ROLLBACK TRAN
   RETURN
END CATCH

BEGIN TRY
-- Opération de mise à jour de Articles
   UPDATE Articles SET Art_Qte = Art_Qte-@pQte
   WHERE Art_Num = @pCodArt
END TRY
BEGIN CATCH
   ROLLBACK TRAN
   RETURN
END TRY
```

COMMIT TRANSACTION

La fonction @@**TRANCOUNT** vaut zéro lorsque aucune transaction n'est ouverte ; une instruction **BEGIN TRAN** incrémente @@**TRANCOUNT** de 1, et une instruction **ROLLBACK TRAN** décrémente @@**TRANCOUNT** de 1.

Dans la plupart des cas, il est préférable de démarrer une transaction avec l'instruction **BEGIN TRANSACTION**. Toutefois, les transactions peuvent démarrer automatiquement avec certaines instructions (mode de transaction implicite), si l'option **SET IMPLICIT_TRANSACTIONS** a été positionnée à ON (par défaut à OFF).

SET IMPLICIT_TRANSACTIONS {ON | OFF}

L'exécution des instructions ALTER TABLE, CREATE, DELETE, GRANT, INSERT, REVOKE, SELECT, UPDATE déclenchent le démarrage de la transaction.

IV VERROUILLAGES DANS SQL SERVER

Lors de transactions concurrentes, des verrous peuvent être posés pour éviter les situations suivantes :

- **.Mise à jour perdue** : une mise à jour peut être perdue lorsqu'une transaction écrase les modifications effectuées par une autre transaction
- **Lecture incorrecte :** Se produit lorsqu'une transaction lit des données non validées provenant d'une autre transaction
- **Lecture non renouvelable :** Se produit lorsque, une transaction devant lire la même ligne plusieurs fois, la ligne est modifiée par une autre transaction et donc produit des valeurs différentes à chaque lecture
- **Lectures fantômes :** Se produit lorsqu'une autre transaction insert une ligne au sein de la transaction en cours

Un **verrou partagé** appliqué à une ressource par une première transaction autorise l'acquisition par une deuxième transaction d'un verrou partagé sur cette ressource, même si la première transaction n'est pas terminée. Un verrou partagé sur une ligne est libéré dès la lecture de la ligne suivante.
SQL Server utilise des **verrous exclusifs** pour les modifications de données (INSERT, UPDTAT, DELETE). Une seule transaction peut acquérir un verrou exclusif sur une ressource ; une transaction ne peut pas acquérir un verrou exclusif sur une ressource tant qu'il existe des verrous partagés ; une transaction ne peut pas acquérir un verrou partagé sur une ressource déjà pourvue d'un verrou exclusif.

SQL Server permet de définir un niveau d'isolement qui protège une transaction vis-à-vis des autres.

SET TRANSACTION ISOLATION LEVEL {READ COMMITTED | READ UNCOMMITTED | REPEATABLE READ | SNAPSHOT | SERIALIZABLE}

Read UnCommitted
Indique à SQL Server de ne pas placer de verrous partagés : une transaction peut lire des données modifiées non encore validées par une autre transaction.
Des lectures incorrectes peuvent se produire.

Read Committed (Option par défaut)
Indique à SQL Server d'utiliser des verrous partagés pendant la lecture : une transaction ne peut pas lire les données modifiées mais non validées d'une autre transaction.
La **lecture incorrecte** ne peut pas se produire.

Repeatable Read
SQL Server place des verrous partagés sur toutes les données lues par chaque instruction de la transaction et les maintient jusqu'à la fin de la transaction : une transaction ne peut pas lire des données modifiées mais pas encore validées par une autre transaction, et ne peut modifier les données lues par la transaction active tant que celle-ci n'est pas terminée.
La **lecture incorrecte** et **la lecture non renouvelable** ne peuvent pas se produire.

Snapshot

Une transaction .n'a pas accès aux modifications de données apportées par une autre transaction : les données vues par la première transaction sont les données antérieures au début de la deuxième transaction.

Serializable

Une transaction ne peut pas lire des données modifiées mais pas encore validées par une autre transaction, et ne peut modifier les données lues par la transaction active tant que celle-ci n'est pas terminée.

Une transaction ne peut pas insérer de nouvelles lignes avec des valeurs de clés comprises dans le groupe de clés lues par des instructions de la transaction active, tant que celle-ci n'est pas terminée

La **lecture incorrecte la lecture non renouvelable et les lectures fantômes** ne peuvent pas se produire.

Le niveau d'isolement spécifie le comportement de verrouillage par défaut de toutes les instructions de la session (connexion).

Plus le niveau d'isolement est élevé, plus les verrrous sont maintenus longtemps et plus ils sont restrictifs.

La commande **DBCC USEROPTIONS** renvoie entre autre, le niveau d'isolement de la session.

Il est possible de définir la durée maximale pendant laquelle SQL Server permet à une transaction d'attendre le déverrouillage d'une ressource bloquée.

La variable globale @@**lock_timeout** donne le temps d'attente défini.

SET LOCK_TIMEOUT *délai_attente* positionne le délai, en millisecondes, avant que ne soit renvoyé un message d'erreur (-1 indique qu'il n'y a pas de délai – par défaut)

Etablissement référent
Marseille Saint Jérôme

Equipe de conception
Elisabeth Cattanéo

Remerciements :

Date de mise à jour 05/05/2008
afpa © Date de dépôt légal mai 08

afpa / Direction de l'Ingénierie13 place du Générale de Gaulle / 93108 Montreuil Cedex
association nationale pour la formation professionnelle des adultes
Ministère des Affaires sociales du Travail et de la Solidarité