Gérer les transactions : les transactions imbriquées avec MS SQL Server

par Frédéric Brouard, alias SQLpro
MVP SQL Server
Expert langage SQL, SGBDR, modélisation de données
Enseigne à l'ISEN Toulon et aux Arts & Métiers
Auteur de :

- SQLpro http://sqlpro.developpez.com/
- "SQL", coll. Synthex, avec C. Soutou, Pearson Education 2005
- "SQL" coll. Développement, Campus Press 2001

Enseigant aux Arts & Métiers et à l'ISEN Toulon

Une grosse difficulté qui attendent les développeurs est de savoir comment piloter les transactions dès lors que celle-ci s'emboitent les unes dans les autres notamment lors des appels de procédures stockées.
C'est ce que l'on appelle les transactions imbriquées.
Cet article présente sommairement la difficulté et le moyen de gérer le plus proprement possible de telles transactions dans le cadre d'un développement recourant généreusement aux procédures stockées et aux transactions...

Ce qu'est..., ce que n'est pas... une transaction

Une transaction est un ensemble de traitements devant être effectué en tout ou rien, en vertu du principe d'atomicité des transactions. Par exemple un virement bancaire d'un compte courant à un compte épargne nécessite une première requête UPDATE pour soutirer l'argent du compte courant et une seconde pour créditer le compte épargne. Si l'une des deux requêtes ne s'effectue pas, alors la base devient incohérente. On dit ainsi que la transaction assure que la base de données part d'un état de cohérence pour arriver dans un autre état de cohérence, les états transitoires, c'est à dire les différentes étapes de la transaction, ne devant jamais être présentés de quelques manière que ce soit, même en cas de panne du système.

Mais que se passe t-il si une transaction démarre à l'intérieur d'une autre transaction ? C'est ce que l'on appelle "transaction imbriquée".
Les transactions imbriquées sont le plus souvent le fait de procédures stockées qui s'appellent les unes des autres afin de fournir un ensemble cohérent de traitement donc chaque partie peut en outre être individuellement appelée.

Le cas est assez classique. On le trouve par exemple lorsque le modèle de données cartographie un objet et que différentes procédures concourent à l'insertion de ses données comme à sa mise à jour. Par exemple une première procédure gère la mise à jour (INSERT / UPDATE / DELETE) d'une personne et appelle une seconde procédure qui gère la mise à jour des adresses relatives à cette personne. D'où deux transactions (une dans chaque procédure) qui fatalement vont s'imbriquées.

Par exemple une procédure stockée 1 démarre une transaction et au milieu de code, alors que la transaction 1 n'est pas finalisée, fait appel à une autre procédure stockée qui elle même encapsule une procédure stockée... Qui valide finalement la transaction ? La procédure appelante ou celle qui est appelée ? Qui annule finalement la transaction ?

Or le principe même d'une transaction imbriquée n'a pas de sens. En effet une transaction est un ensemble cohérent. Imaginons le scénario suivant :

```
BEGIN TRANSACTION A

... code a1 ...

   BEGIN TRANSACTION B

   ... code b ...

   ROLLBACK TRANSACTION B

... code a2 ...
```

```
COMMIT TRANSACTION A
```

La transaction B valide les parties de code a1 et a2, mais le code b étant annulé, la transaction A est clairement incohérente. C'est pourquoi dans le principe **les transactions imbriquées ne sont pas possibles** !
En fait il n'y a donc jamais qu'une seule transaction. Et c'est toujours la première...

Modèle de transactions imbriquées

Dès lors deux modèles de "pseudo" transactions imbriquées sont possibles : le modèle symétrique et le modèle asymétrique.

Dans le modèle symétrique, le premier BEGIN TRANSACTION commence la vraie seule transaction. Chaque fois qu'un nouveau BEGIN TRANSACTION est rencontré dans le code, la commande est ignorée, mais un compteur de transaction est incrémenté de 1. Chaque fois qu'un COMMIT ou ROLLBACK est rencontré, ce même compteur est décrémenté de 1 et la commande n'a pas d'effet. Si le compteur est à zéro, alors le COMMIT ou ROLLBACK rencontré est réellement exécuté. Cela peut se résumé par le script suivant :

```
BEGIN TRANSACTION A

... code a1 ...

   BEGIN TRANSACTION B -- code ignoré, compteur "tran" à 1

   ... code b ...

   ROLLBACK TRANSACTION B -- code ignoré, compteur "tran" à 0

... code a2 ...

COMMIT TRANSACTION A
```

Ainsi comme on le voit, tout le code de cette procédure est exécuté.

Mais ce n'est pas le comportement adopté par MS SQL Server... En effet, ce SGBDR se base sur le modèle asymétrique, finalement bien plus fin !

Le modèle asymétrique de transaction imbriqué

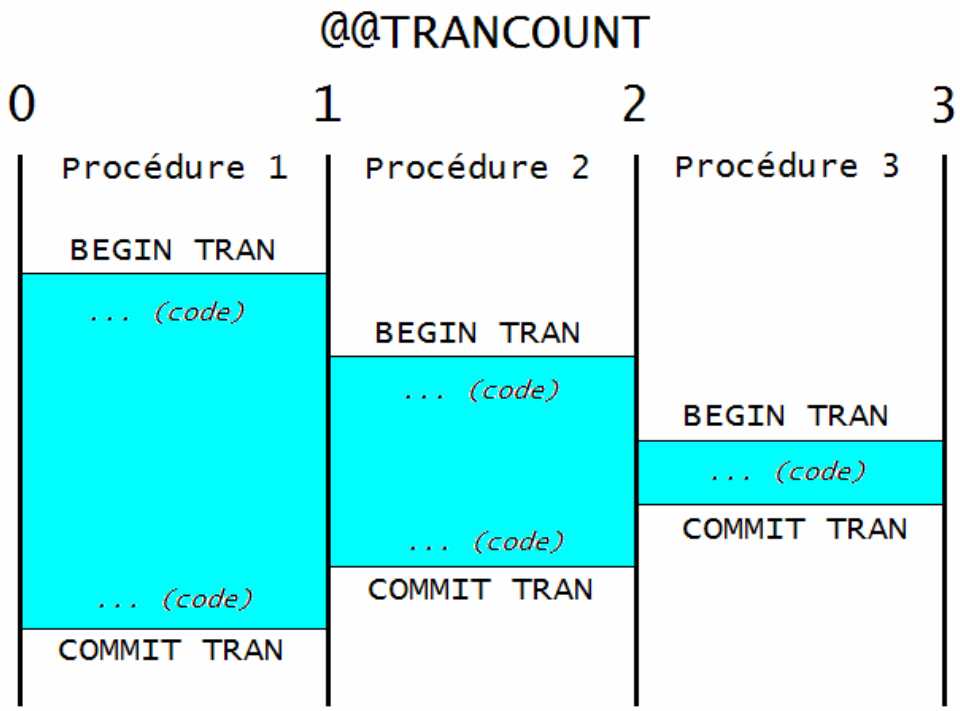
Le principe du modèle asymétrique de transactions imbriquées est simple, mais sa mise en œuvre réserve quelques surprises !

Voici les règles de base :

1) il n'y a jamais qu'une seule transaction
2) Le premier BEGIN TRANSACTION rencontré démarre la transaction

3) tout autre BEGIN TRANSACTION que le premier ne fait qu'incrémenter le compteur de session @@TRANCOUNT
4) le premier ROLLBACK TRANSACTION rencontré annule la transaction
5) chaque COMMIT TRANSACTION décrémente le compteur de session @@TRANCOUNT de 1 et si ce compteur vaut 0 alors la transaction est finalement validée.

Voici ce qui se passe lorsque des transactions imbriquées réussissent :

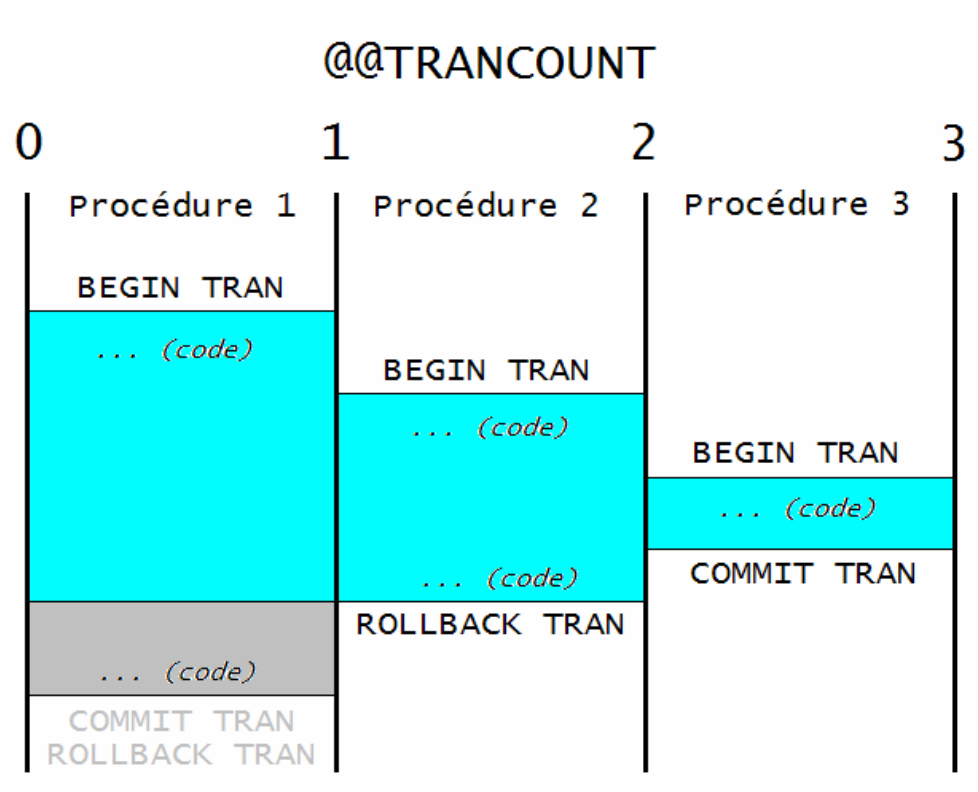


Dans cet exemple, la procédure 1 démarre une transaction avec un BEGIN TRANSACTION et met le compteur @@TRANCOUNT à 1, puis appelle la procédure 2 qui, voyant qu'une transaction est déjà démarrée, ne fait que mettre le compteur @@TRANCOUT à 2, puis appelle la procédure 3 qui, voyant qu'une transaction est déjà démarrée, ne fait que mettre le compteur @@TRANCOUT à 3. Cette dernière transaction réussie et ne fait que décrémenter le compteur @@RANCOUNT qui passe de 3 à 2 puis revient en procédure 2, qui elle même réussit aussi et ne fait que passer le compteur @@TRANCOUNT de 2 à 1. Enfin la procédure 1 fait le commit final qui fait passer @@TRANCOUNT de 1 à 0 et génère réellement le COMMIT !

En tout et pour tout il n'y a eût qu'un seul BEGIN TRANSACTION et un seul COMMIT TRANSACTION.

La notion même de transaction imbriquée n'existe donc pas...

Que se passe t-il si une transaction interne génère un rollback ?



En fait dans ce cas le ROLLBACK est immédiatement exécuté et le compteur @@TRANCOUNT passe à zéro.
Si jamais le code dans procédure 1 passe par le COMMIT, alors le système ne s'y retrouve plus et génère un message d'erreur du genre : *Le compte des transactions après EXECUTE indique qu'il manque une instruction COMMIT ou ROLLBACK TRANSACTION*

Démonstration :

```
-- création d'une table test pour notre transaction
IF EXISTS (SELECT *
           FROM    INFORMATION_SCHEMA.TABLES
           WHERE   TABLE_SCHEMA = 'dbo'
             AND   TABLE_NAME = 'T_TRN')
   DROP TABLE T_TRN
GO

-- table avec une contrainte de validité
CREATE TABLE T_TRN
(N INT CHECK (N >= 0))
GO


-- création d'une procédure stockée de test de transaction imbriquée
IF EXISTS (SELECT *
           FROM    INFORMATION_SCHEMA.ROUTINES
           WHERE   ROUTINE_SCHEMA = 'dbo'
             AND   ROUTINE_NAME = 'P_TRN_INTERNE')
   DROP PROCEDURE P_TRN_INTERNE
GO

CREATE PROCEDURE P_TRN_INTERNE
AS
```

```
DECLARE @ERROR INT, @ROWCOUNT INT

BEGIN TRANSACTION

-- insertion invalide : elle doit déclencher le ROLLBACK
INSERT INTO T_TRN VALUES (-4)
SELECT @ERROR = @@ERROR, @ROWCOUNT = @@ROWCOUNT
IF @ERROR <> 0 OR @ROWCOUNT = 0
BEGIN
   RAISERROR('Procédure P_TRN_INTERNE : Erreur à l''insertion', 16, 1)
   GOTO LBL_ERROR
END

COMMIT TRANSACTION

RETURN (0)


LBL_ERROR:

IF @@TRANCOUNT > 1
   COMMIT TRANSACTION
IF @@ROWCOUNT = 1
   ROLLBACK TRANSACTION
RETURN (-1)

GO


-- création d'une procédure stockée de test de transaction imbriquée
IF EXISTS (SELECT *
           FROM    INFORMATION_SCHEMA.ROUTINES
           WHERE   ROUTINE_SCHEMA = 'dbo'
             AND   ROUTINE_NAME = 'P_TRN_EXTERNE')
   DROP PROCEDURE P_TRN_EXTERNE
GO


CREATE PROCEDURE P_TRN_EXTERNE
AS

DECLARE @ERROR INT, @ROWCOUNT INT, @RETVAL INT

BEGIN TRANSACTION

-- insertion valide
INSERT INTO T_TRN VALUES (33)
SELECT @ERROR = @@ERROR, @ROWCOUNT = @@ROWCOUNT
IF @ERROR <> 0 OR @ROWCOUNT = 0
BEGIN
   RAISERROR('Procédure P_TRN_EXTERNE : Erreur à l''insertion', 16, 1)
   GOTO LBL_ERROR
END

EXEC @RETVAL = P_TRN_INTERNE
SELECT @ERROR = @@ERROR, @ROWCOUNT = @@ROWCOUNT

IF @RETVAL = -1 -- la transaction a été pseudo validée mais elle doit être
annulée
BEGIN
   RAISERROR('Procédure P_TRN_EXTERNE : Erreur à l''appel de la procédure
P_TRN_INTERNE', 16, 1)
   GOTO LBL_ERROR
END

IF @ERROR <> 0 OR @@ROWCOUNT = 0
   GOTO LBL_ERROR

COMMIT TRANSACTION
```

```
RETURN (0)


LBL_ERROR:

IF @@TRANCOUNT > 1
   COMMIT TRANSACTION
IF @@ROWCOUNT = 1
   ROLLBACK TRANSACTION
RETURN (-1)

GO
-- exécution teste
EXEC P_TRN_EXTERNE
GO

-- a l'issu de cet exécution aucune ligne ne doit avoir été inséré :
SELECT * FROM T_TRAN
GO
```

Piloter génériquement des transactions imbriquées

Si vous voulez piloter proprement des transactions qui s'emboîtent dans d'autres transactions notamment lorsque vous faîtes appel à des procédures stockées qui s'imbriques les unes dans les autres il faut gérer le COMMIT ou le ROLLBACK en tenant compte de la valeur du compteur @@TRANCOUNT.

Voici comment finaliser proprement une transaction quelque soit le contexte transactionnel :

```
-- partie à rajouter à TOUTES les procédures (finalisation) :

-- succès
COMMIT TRANSACTION
RETURN (0)

-- échec
LBL_ERROR:
IF @@TRANCOUNT > 1
   COMMIT TRANSACTION
IF @@ROWCOUNT = 1
   ROLLBACK TRANSACTION
RETURN (-1)
```

Si vous avez besoin de rétablir le niveau d'isolation par défaut :

```
...
DECLARE @RETVAL INT

...

-- succès
COMMIT TRANSACTION
SET @RETVAL = 0
GOTO RESUME

-- échec
LBL_ERROR:
IF @@TRANCOUNT > 1
```

```
   COMMIT TRANSACTION
IF @@ROWCOUNT = 1
   ROLLBACK TRANSACTION
SET @RETVAL = -1

LBL_RESUME:
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
RETURN @RETVAL
```