Gestion des transactions avec SQL Server

par Baptiste Wicht (home)

Date de publication : Le 15 Décembre 2006

Cet article vous apprendra à garantir/préserver l'intégrité de vos données lors de l'exécution d'un ensemble de requêtes sur votre base de données.

I - Introduction

Vous avez souvent du effectuer une série de requêtes sur une base de données et vous connaissez les risques que vous encourez si l'une d'entre elles échoue et qu'une autre réussit. Vous risquez d'avoir des données incohérentes et de perdre du temps ensuite pour trouver d'ou vient le problème.

Cet article va vous donnez une manière de faire très simple pour sécuriser vos requêtes SQL-Server.

II - Sécurisation

II-A - Introduction

Pour plus de clarté, je vais prendre un exemple concret : admettons que dans une base de données, nous voulions modifier des états civils et en insérer un nouveau et que de plus le texte soit localisé donc que pour un état civil nous avons 3 valeurs texte.

Voici les scripts pour la création des tables et l'insertion des valeurs :

```
CREATE TABLE T_TEXTES (
 ID INT PRIMARY KEY NOT NULL,
 LANGUE VARCHAR(5) NOT NULL,
 TEXTE VARCHAR(255) NOT NULL,
 VALEUR_PARAMETRE INT(11) NOT NULL
)

CREATE TABLE T_VALEUR_PARAMETRE (
 ID INT PRIMARY KEY NOT NULL,
 PARAM_CODE VARCHAR(255) NOT NULL,
 OFFICE INT NOT NULL,
 PARAMETRE INT NULL
)

INSERT INTO T_TEXTES(ID, TEXTE, LANGUE, VALEUR_PARAMETRE) VALUES(1, 'Divorcé', 'fr', 334)
INSERT INTO T_TEXTES(ID, TEXTE, LANGUE, VALEUR_PARAMETRE) VALUES(2, 'IT:Divorcé', 'it', 334)
INSERT INTO T_TEXTES(ID, TEXTE, LANGUE, VALEUR_PARAMETRE) VALUES(3, 'Marié', 'fr', 334)
INSERT INTO T_TEXTES(ID, TEXTE, LANGUE, VALEUR_PARAMETRE) VALUES(4, 'IT:Marié', 'it', 334)
INSERT INTO T_TEXTES(ID, TEXTE, LANGUE, VALEUR_PARAMETRE) VALUES(5, 'Veuf', 'fr', 334)
INSERT INTO T_TEXTES(ID, TEXTE, LANGUE, VALEUR_PARAMETRE) VALUES(6, 'IT:Veuf', 'it', 334)
INSERT INTO T_TEXTES(ID, TEXTE, LANGUE, VALEUR_PARAMETRE) VALUES(7, 'Séparé', 'fr', 334)
INSERT INTO T_TEXTES(ID, TEXTE, LANGUE, VALEUR_PARAMETRE) VALUES(8, 'IT:Séparé', 'it', 334)
```

Tout de suite, vous feriez quelque chose dans ce goût-là :

```
DECLARE
 @ID_INSERTION NUMERIC(19,0) -- On déclare une variable numérique destinée à contenir l'id inséré

UPDATE T_TEXTES SET TEXTE='Divorcé(e)' WHERE ID = '1'
UPDATE T_TEXTES SET TEXTE='IT:Divorcé(e)' WHERE ID = '2'
UPDATE T_TEXTES SET TEXTE='Marié(e)' WHERE ID = '3'
UPDATE T_TEXTES SET TEXTE='IT:Marié(e)' WHERE ID = '4'
UPDATE T_TEXTES SET TEXTE='Veuf(ve)' WHERE ID = '5'
UPDATE T_TEXTES SET TEXTE='IT:Veuf(ve)' WHERE ID = '6'
UPDATE T_TEXTES SET TEXTE='Séparé(e)' WHERE ID = '7'
UPDATE T_TEXTES SET TEXTE='IT:Séparé(e)' WHERE ID = '8'

INSERT INTO T_VALEUR_PARAMETRE (PARAM_CODE, OFFICE, PARAMETRE)
VALUES ('etatCivil.values.pacs.value', '6', '99')

SET @ID_INSERTION =
 @@identity --On récupère la valeur du dernier id inséré et on le stocke dans notre variable

INSERT INTO T_TEXTES (TEXTE, LANGUE, VALEUR_PARAMETRE)
VALUES('Lié(e) par un partenariat enregistré','fr', @ID_INSERTION)
INSERT INTO T_TEXTES (TEXTE, LANGUE, VALEUR_PARAMETRE)
VALUES('IT:Lié(e) par un partenariat enregistré','it', @ID_INSERTION)
INSERT INTO T_TEXTES (TEXTE, LANGUE, VALEUR_PARAMETRE)
VALUES('Verbunden durch eine eingetragene Partnerschaft','de', @ID_INSERTION)
```

Bien sûr, ce code fonctionne, mais si par exemple la première requête d'insertion échoue, vous aurez les 3 valeurs suivantes qui ne feront référence à rien du tout donc incohérentes...

Cela est dû au fait que SQL-Server fonctionne par défaut en mode AUTOCOMMIT et que toute requête est automatiquement validé dès son exécution. C'est pratique dans le cas d'une seule requête, mais quand on doit envoyer un lot de requêtes, ce n'est pas pratique, c'est pourquoi nous allons pallier à ce problème.

II-B - Transaction

Une transaction est un ensemble de requêtes que l'on regroupe en une seule unité logique de travail qui pourra ensuite être, soit validée, soit annulée.

La première chose à faire dans ce cas-là est donc d'encapsuler vos requêtes dans une transaction. On va donc faire démarrer une transaction au début et la committer à la fin :

```
BEGIN TRANSACTION changement_etat_civil --On démarre une transaction et on lui donne un nom

--Vos requêtes

COMMIT TRANSACTION
 changement_etat_civil --On commit cette transaction, c'est à dire qu'on valide ses modifications
```

Mais cela ne change rien au problème, en cas d'erreur, l'ensemble est tout de même committé et nous avons toujours le risque d'avoir des données incohérentes.

II-C - Détection des erreurs

Nous allons donc ajouter une partie de gestion d'erreurs à notre code. Pour récupérer une erreur, il suffit d'employer la variable prédéfini @@ERROR qui contient l'erreur pour la dernière requête effectuée. Nous allons donc déclarer une variable @errors pour stocker les différentes erreurs récupérées lors de l'exécution :

```
BEGIN TRANSACTION changement_etat_civil --On démarre une transaction et on lui donne un nom

DECLARE @errors INT --On déclare une variable qui sera destiné à accueillir nos erreurs
DECLARE
 @ID_INSERTION NUMERIC(19,0) -- On déclare une variable numérique destinée à contenir l'id inséré

UPDATE T_TEXTES SET TEXTE='Divorcé(e)' WHERE ID = '1'
SET @error = @error + @@error
UPDATE T_TEXTES SET TEXTE='IT:Divorcé(e)' WHERE ID = '2'
SET @error = @error + @@error

UPDATE T_TEXTES SET TEXTE='Marié(e)' WHERE ID = '3'
SET @error = @error + @@error

UPDATE T_TEXTES SET TEXTE='IT:Marié(e)' WHERE ID = '4'
SET @error = @error + @@error

UPDATE T_TEXTES SET TEXTE='Veuf(ve)' WHERE ID = '5'
SET @error = @error + @@error

UPDATE T_TEXTES SET TEXTE='IT:Veuf(ve)' WHERE ID = '6'
SET @error = @error + @@error

UPDATE T_TEXTES SET TEXTE='Séparé(e)' WHERE ID = '7'
SET @error = @error + @@error

UPDATE T_TEXTES SET TEXTE='IT:Séparé(e)' WHERE ID = '8'
SET @error = @error + @@error

INSERT INTO T_VALEUR_PARAMETRE (version, PARAM_CODE, OFFICE, PARAMETRE)
VALUES (GetDate() 'etatCivil.values.pacs.value', '6', '99')
SET @errors = @errors +
 @@ERROR --On additionne l'erreur liée à la dernière requête SQL dans notre variable
```

```
SET @errors = @errors +
 @@ERROR --On additionne l'erreur liée à la dernière requête SQL dans notre variable
SET @ID_INSERTION =
 @@identity --On récupère la valeur du dernier id inséré et on le stocke dans notre variable

INSERT INTO T_TEXTES (TEXTE, LANGUE, VALEUR_PARAMETRE)
VALUES('Lié(e) par un partenariat enregistré','fr', @ID_INSERTION)

SET @errors = @errors +
 @@ERROR --On additionne l'erreur liée à la dernière requête SQL dans notre variable

INSERT INTO T_TEXTES (TEXTE, LANGUE, VALEUR_PARAMETRE)
VALUES('IT:Lié(e) par un partenariat enregistré','it', @ID_INSERTION)

SET @errors = @errors +
 @@ERROR --On additionne l'erreur liée à la dernière requête SQL dans notre variable

INSERT INTO T_TEXTES (TEXTE, LANGUE, VALEUR_PARAMETRE)
VALUES('Verbunden durch eine eingetragene Partnerschaft','de', @ID_INSERTION)

SET @errors = @errors +
 @@ERROR --On additionne l'erreur liée à la dernière requête SQL dans notre variable

COMMIT TRANSACTION
 changement_etat_civil --On commit cette transaction, c'est à dire qu'on valide ses modifications
```

Vous pouvez aussi afficher les erreurs avec :

```
PRINT 'Statut de l'erreur : ' + CAST(@errors AS VARCHAR(10)) --On affiche le statut de l'erreur casté
 sous forme de caractère
```

juste avant le commit.

II-D - Gestion des erreurs

Bon, c'est bien beau, vous me direz, on voit les erreurs, mais en cas d'erreurs notre base est toujours incohérente. Bien, ca prouve que vous suivez. Une astuce pour cela est de déclarer notre variable @errors à 1, comme ça, si à la fin de l'exécution des requêtes, elle n'est plus à un, on sait qu'il y a eu une erreur et on peut faire quelque chose.

```
DECLARE @errors INT

SET @errors = 0 --On déclare notre variable à 0

--Vos requêtes
```

On va maintenant contrôler s'il y a eu ou non des erreurs et s'il y en a eu, on va annuler toutes les opérations faites dans la transaction en utilisant la commande ROLLBACK TRANSACTION :

```
IF @errors = 0 --Si errors est égale à 0, donc s'il n'y a eu aucune erreur
 COMMIT TRANSACTION changement_etat_civil -- On commit la transaction
ELSE --S'il y a eu des erreurs
 ROLLBACK TRANSACTION changement_etat_civil --On annule tous les changements de cette transaction
```

Voilà, vous avez maintenant un code qui vous affiche les erreurs qui sont arrivées au cours du programme et qui annule toutes vos modifications en cas de problème, comme ça, plus de risques d'incohérence de données en cas de problème lors de l'exécution d'une requête.

II-E - Points d'enregistrements

Une autre chose qu'il est utile de connaître est le fait qu'on peut insérer des savepoint dans une transaction, c'est à dire un point d'enregistrement, sur lequel on peut se baser pour faire un RollBack. Ainsi, on n'a pas besoin de faire une annulation sur l'intégralité de la transaction, si on a deux parties bien distinctes dans notre script, on peut intercaler un point d'enregistrement entre les 2. Si la première partie s'est déroulée sans problèmes, mais que la deuxième partie s'est mal passé, on faire un rollback jusqu'à notre point d'enregistrement puis un commit. Ainsi, tout ce qui s'est bien passé est validé et le reste est annulé.

Pour sauvegarder un point d'enregistrement, il vous suffit de faire ceci :

```
SAVE TRANSACTION nom_du_savepoint --On sauvegarde un point d'enregistrement pour la transaction
```

Et si ensuite, vous voulez revenir à cet état, il vous suffit de faire :

```
ROLLBACK TRANSACTION
 nom_du_savepoint --On annule toutes les modifications de cette transaction jusqu'au point d'enregistrement
```

II-F - Le script final

```
BEGIN TRANSACTION changement_etat_civil --On démarre une transaction et on lui donne un nom

DECLARE @errors INT --On déclare une variable qui sera destiné à accueillir nos erreurs
DECLARE
 @ID_INSERTION NUMERIC(19,0) -- On déclare une variable numérique destinée à contenir l'id inséré

SET @errors = 0

UPDATE T_TEXTES SET TEXTE='Divorcé(e)' WHERE ID = '1'
SET @error = @error + @@error
UPDATE T_TEXTES SET TEXTE='IT:Divorcé(e)' WHERE ID = '2'
SET @error = @error + @@error

UPDATE T_TEXTES SET TEXTE='Marié(e)' WHERE ID = '3'
SET @error = @error + @@error

UPDATE T_TEXTES SET TEXTE='IT:Marié(e)' WHERE ID = '4'
SET @error = @error + @@error

UPDATE T_TEXTES SET TEXTE='Veuf(ve)' WHERE ID = '5'
SET @error = @error + @@error

UPDATE T_TEXTES SET TEXTE='IT:Veuf(ve)' WHERE ID = '6'
SET @error = @error + @@error

UPDATE T_TEXTES SET TEXTE='Séparé(e)' WHERE ID = '7'
SET @error = @error + @@error

UPDATE T_TEXTES SET TEXTE='IT:Séparé(e)' WHERE ID = '8'
SET @error = @error + @@error

INSERT INTO T_VALEUR_PARAMETRE (version, PARAM_CODE, OFFICE, PARAMETRE)
VALUES (GetDate() 'etatCivil.values.pacs.value', '6', '99')
SET @errors = @errors +
 @@ERROR --On additionne l'erreur liée à la dernière requête SQL dans notre variable

SET @errors = @errors +
 @@ERROR --On additionne l'erreur liée à la dernière requête SQL dans notre variable
SET @ID_INSERTION =
 @@identity --On récupère la valeur du dernier id inséré et on le stocke dans notre variable

INSERT INTO T_TEXTES (TEXTE, LANGUE, VALEUR_PARAMETRE)
```

```
VALUES('Lié(e) par un partenariat enregistré','fr', @ID_INSERTION)

SET @errors = @errors +
 @@ERROR --On additionne l'erreur liée à la dernière requête SQL dans notre variable

INSERT INTO T_TEXTES (TEXTE, LANGUE, VALEUR_PARAMETRE)
VALUES('IT:Lié(e) par un partenariat enregistré','it', @ID_INSERTION)

SET @errors = @errors +
 @@ERROR --On additionne l'erreur liée à la dernière requête SQL dans notre variable

INSERT INTO T_TEXTES (TEXTE, LANGUE, VALEUR_PARAMETRE)
VALUES('Verbunden durch eine eingetragene Partnerschaft','de', @ID_INSERTION)

SET @errors = @errors +
 @@ERROR --On additionne l'erreur liée à la dernière requête SQL dans notre variable

PRINT 'Statut de l'erreur : ' + CAST(@errors AS VARCHAR(10)) --On affiche le statut de l'erreur casté
 sous forme de caractère

IF @errors = 0 --Si errors est égale à 0, donc s'il n'y a eu aucune erreur
 COMMIT TRANSACTION changement_etat_civil -- On commit la transaction
ELSE --S'il y a eu des erreurs
 ROLLBACK TRANSACTION changement_etat_civil --On annule tous les changements de cette transaction
```

III - Conclusion

En conclusion, il n'est vraiment pas difficile de sécuriser du code SQL-Server, il suffit d'un peu de rigueur et vous aurez ainsi supprimé une bonne part de surprise et aurez des codes plus sûrs à exécutez.

IV - Remerciements

Je tiens à remercier **Xo** pour ses corrections.