# Niveau d'isolation et anomalies transactionnelles.

*Peu connu et souvent mal utilisés, les niveaux d'isolation des transactions permettent de rendre plus ou moins étanches (ou poreuses) les transactions s'exécutant de manière simultanées. Cet article propose de voyager au coeurs des problèmes de concurrences de traitements dans le monde des bases de données relationnelles et d'aborder en toute sérénité ce que sont les anomalies transactionnelles et comment les différents niveaux d'isolation de la norme SQL prévoit leur empêchement.*


Par Frédéric Brouard - MVP SQL Server
Expert SQL et SGBDR, Auteur de :

- SQL, Développement, Campus Press 2001
- SQL, collection Synthex, Pearson Education 2005, co écrit avec Christian Soutou
- http://sqlpro.developpez.com (site de ressources sur le langage SQL et les SGBDR)
- Enseigant aux Arts & Métiers et à l'ISEN Toulon



***Démonstration avec SQL Server 2005***

Au pire, on pourrait considérer que toute opération atomique devrait impérativement placer la resource visée par la transaction en accès exclusif. Mais ce mode de fonctionnement ne serait pas sain et très surtout peu performant. En effet, en imaginant une très grande table, il ne serait pas judicieux de bloquer l'accès à toutes les lignes de la table alors que l'on met à jour une seule ligne de cette table dans une transaction, tandis qu'une autre transaction supprime ou modifie un jeu de ligne situé ailleurs. C'est pourquoi différentes granularités de verrouillage existent ainsi que différents modes.

La granularité d'un verrou pose le problème de son étendue : la ligne, la page de données (donc généralement plusieurs lignes), l'extension (en fait 8 pages), la table...
Le mode de verouillage est lié au traitement considéré. Ainsi une lecture oblige à un verrou de lecture qui empêche toute modification des données vérouillées mais n'empêche pas que d'autres transactions lisent une partie ou la totalité des mêmes données. A contrario un verrou d'écriture empêche la pose de tout autre verrou qu'il soit de lecture ou d'écriture.

Voici par exemple les différents mode de verrouillage que propose MS SQL Server 2005 :

| Mode | Description |
|---|---|
| Partagé (S) | Utilisé pour les opérations de lecture qui n'effectuent aucune mise à jour. (exemple : SELECT). |
| Mise à jour (U) | Utilisé pour les mises à jour par UPDATE. Empêche que plusieurs sessions lisent, verrouillent et mettent à jour les mêmes ressources ultérieurement. |
| Exclusif (X) | Utilisé pour les mises à jour par INSERT, UPDATE ou DELETE. Empêche des mises à jour multiples sur la même ressource au même moment. |
| Intentionnel | Permet d'établir une hiérarchie de verrouillage. Les types de verrouillage intentionnels sont les suivants : partage intentionnel (IS), exclusion intentionnelle (IX), partage intentionnel exclusif (SIX). modification intentionnelle (UI), partagé et UI simultané (SIU), exclusif et exclusion intentionnelle avec update. |
| Schéma | Utilisé lors de l'exécution d'une opération de modification du schéma d'une table (par exemple ALTER). Les types de verrouillage de schéma sont les suivants : modification de schéma (Sch-S) et stabilité de schéma (Sch-M). |
| Mise à jour en bloc (BU) | Utilisé lors de la copie en bloc de données (BULK INSERT) dans une table avec l'indicateur TABLOCK spécifié. |
| Verrou de clé | Protège la plage de lignes lue par une requête lorsque le niveau d'isolation des transactions SERIALIZABLE est utilisé. Garantit qu'aucune autre transaction ne peut insérer des lignes susceptibles de répondre aux requêtes de la transaction sérialisable si ces dernières étaient réexécutées. RS : Row Shared, RU : Row Update, RI : Row Intent, RX : Row eXclusive. |

Mais ce n'est pas en principe au développeur d'indiquer le verrou qu'il entend utiliser[1], c'est au moniteur de verrouillage de faire ce travail au mieux, en tenant compte de la concurrence des transactions, de la facilité d'acquérir tel ou tel type de verrou ou encore de l'existence préalable d'autres verrous. En d'autre termes, il est rare que la pose explicite de verrous fasse plus de bien que de mal, car elle oblige à statifier l'état du verrou, alors que le moniteur peut le faire dynamiquement en choisissant à bon escient le verrou le mieux adpaté compte tenu des circonstances ou momment ou le traitement s'exécute.

C'est pour cela que SQL propose d'utiliser un niveau logique de gestion de la concurrence, le fameux niveau d'isolation des transactions. Ces niveaux logiques permettent de s'affranchir de différentes anomalies transactionelles qui peuvent être induits par la nature des verrous sous jacents.

Mais au fait, qu'est-ce qu'une anomalie transactionnelle ? SQL (le langage) considère principalement 3 anomalies transactionnelle :

- la lecture de données inconsistente (ou lecture sale);
- la lecture non répétitible de données;

[1] Même si cela est possible dans les requêtes SQL Server via des tags introduits par l'option WITH.

- la lecture de données "fantômes" c'est à dire apparaissant au beau milieu d'un traitement.

Voici un tableau résumant les différents niveaux d'isolation et les anomalies transactionnelles possible ou non :

| Niveau d'isolation | | | | |
|---|---|---|---|---|
| **Ordre** | **Nom** | **lectures inconsistentes** | **Lectures non répétitibles** | **Lecture fantôme** |
| 0 | READ UNCOMMITTED (lecture de données non vaidées) | **Oui** | **Oui** | **Oui** |
| 1 | READ COMMITTED (lecture de données validées) | Non | **Oui** | **Oui** |
| 2 | REPEATABLE READ (lectures répétitible) | Non | Non | **Oui** |
| 3 | SNAPSHOT (lecture de copie de données) | Non | Non | Non |
| 3 | SERIALIZABLE (mise en série de la transaction) | Non | Non | Non |

**NOTA** : le niveau d'isolation SNAPSHOT propre à SQL Server et présent aussi dans Oracle ne fait pas partie de la norme SQL.

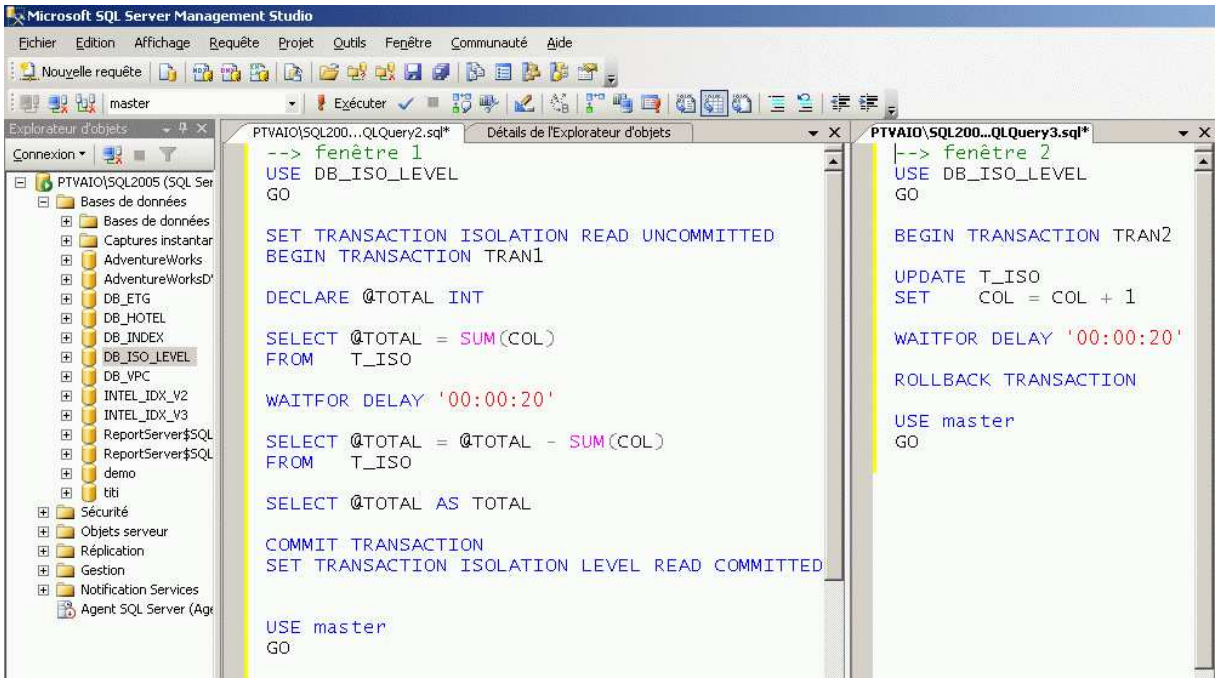
Pour mieux comprendre ces anomalies et leur erradication à l'aide des différents niveaux d'isolation, rien ne vaut un exemple concret. Cet exemple consiste à calculer par deux fois une somme de nombres dans la même table et d'en faire la soustraction. En principe un tel calcul devrait systématiquement retourner zéro... Et pourtant !

Voyons tout d'abord notre jeu d'essais. Vous devrez le rejouer à chaque nouvelle tentative.

```
--> le script de la création de la base qui servira d'exemple
USE master
GO

IF EXISTS (SELECT *
           FROM   master.sys.databases
           WHERE  name = 'DB_ISO_LEVEL')
   DROP DATABASE DB_ISO_LEVEL
GO

CREATE DATABASE DB_ISO_LEVEL
GO

USE DB_ISO_LEVEL
GO

CREATE TABLE T_ISO ( COL INT)
GO

INSERT INTO T_ISO VALUES (1)
INSERT INTO T_ISO VALUES (2)
INSERT INTO T_ISO VALUES (3)
GO

USE master
GO
```

*Script n° 1*

Par la suite, pour jouer les scripts SQL qui mettent en évidence les anomalies évoquées en titre, vous devrez placer le code à exécuter dans deux fenêtres. Lancez ensuite l'exécution de la première fenêtre puis de la seconde dans la foulée. Observez alors les résultats…




Dans les scripts ci dessous qui simulent une concurrence, nous avons sciemment utilisé l'instruction WAITFOR qui permet de faire attendre artificiellement le traitement.

### 1 - S'affranchir des anomalies de lectures « sales » (*lectures imporpore, lectures non validées…*)

Ce type d'anomalie intervient si la transaction courante s'exécute avec un niveau d'isolation READ UNCOMMITTED, c'est-à-dire sans aucune pose de verrou d'aucune sorte lors des lectures.

```
--> fenêtre 1
USE DB_ISO_LEVEL
GO

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
BEGIN TRANSACTION TRAN1

DECLARE @TOTAL INT

SELECT @TOTAL = SUM(COL)
FROM   T_ISO

WAITFOR DELAY '00:00:20'
```

```
SELECT @TOTAL = @TOTAL - SUM(COL)
FROM   T_ISO

SELECT @TOTAL AS TOTAL

COMMIT TRANSACTION
SET TRANSACTION ISOLATION LEVEL READ COMMITTED

USE master
GO
```

```
--> fenêtre 2
USE DB_ISO_LEVEL
GO

BEGIN TRANSACTION TRAN2

UPDATE T_ISO
SET    COL = COL + 1

WAITFOR DELAY '00:00:20'

ROLLBACK TRANSACTION

USE master
GO
```

*Script n° 2*

Le résultat de la transaction 1 affiche en principe -3.

Pour s'affranchir des anomalies de lectures « sales » il suffit de placer la transaction 1 au niveau d'isolation READ COMMITTED. Cela met en oeuvre des verrous sur les lectures empêchant que la modification des lignes ne soit commise pendant la lecture.

### 2 - S'affranchir des anomalies de lectures « non répétitibles »

Ce type d'anomalie intervient si la transaction courante s'exécute avec un niveau d'isolation READ COMMITTED :

*NOTA : pour la démonstration, n'oubliez pas de reconstruire la base originale à l'aide du script n°1*

```
--> fenêtre 1
USE DB_ISO_LEVEL
GO

SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION TRAN1

DECLARE @TOTAL INT

SELECT @TOTAL = SUM(COL)
```

```
FROM   T_ISO

WAITFOR DELAY '00:00:20'

SELECT @TOTAL = @TOTAL - SUM(COL)
FROM   T_ISO

SELECT @TOTAL AS TOTAL

COMMIT TRANSACTION
SET TRANSACTION ISOLATION LEVEL READ COMMITTED

USE master
GO
```

```
--> fenêtre 2
USE DB_ISO_LEVEL
GO

BEGIN TRANSACTION TRAN2

UPDATE T_ISO
SET    COL = COL + 1

WAITFOR DELAY '00:00:20'

COMMIT TRANSACTION

USE master
GO
```

*Script n° 3*

A nouveau le résultat de la transaction 1 affiche -3. Que s'est il passé ? Des verrous ont bien été posé sur les 3 lignes qui figurent au départ dans la table. Mais les verrous n'ont été posé que le temps que chaque requête et n'ont pas été maintenus jusqu'à la fin de la transaction. Tant est si bien qu'entre les deux SELECT l'UPDATE à produit son effet.

Pour s'affranchir des anomalies de lectures « non répétitibles » il suffit de placer la transaction 1 au niveau d'isolation REPEATABLE READ. Ceci pose des verous sur les lignes lues et maintient les verrous jusqu'à la fin de la transaction.

### 3 - S'affranchir des anomalies de lectures « fantômes »

*NOTA : pour la démonstration, n'oubliez pas de reconstruire la base originale à l'aide du script n°1*

Ce type d'anomalie intervient si la transaction courante s'exécute avec un niveau d'isolation REPEATABLE READ :

```
--> fenêtre 1
```

```
USE DB_ISO_LEVEL
GO

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION TRAN1

DECLARE @TOTAL INT

SELECT @TOTAL = SUM(COL)
FROM   T_ISO

WAITFOR DELAY '00:00:20'

SELECT @TOTAL = @TOTAL - SUM(COL)
FROM   T_ISO

SELECT @TOTAL AS TOTAL

COMMIT TRANSACTION
SET TRANSACTION ISOLATION LEVEL READ COMMITTED

USE master
GO
```

```
--> fenêtre 2
USE DB_ISO_LEVEL
GO

BEGIN TRANSACTION TRAN2

INSERT INTO T_ISO VALUES (4)

WAITFOR DELAY '00:00:20'

COMMIT TRANSACTION

USE master
GO
```

*Script n° 4*

Pour s'affranchir des anomalies de lectures « fantôme » il suffit de placer la transaction 1 au niveau d'isolation SERIALIZABLE[2]. Ceci pose un verrou exclusif sur la table et le maintient jusqu'à la fin de la transaction.

Un moyen moins couteux en termes de verrouillage est proposé depuis la version 2005 de SQL Server. Il s'agit d'utiliser le niveau d'isolation **SNAPSHOT** qui réalise une copie des données à traiter pour la durée de la transaction.

[2] En fait la norme SQL affirme que le mode SERIALIZABLE empêche toute anomalie transactionnelle de quelque nature que ce soit. Il existe en effet d'autres types d'anomalies transactionnelles moins faciles à mettre en évidences, et ces anomalies là sont aussi éradiquées par le mode SERIALIZABLE.

**Attention** : une base de données est créée par défaut sans la possibilité d'utiliser ce niveau d'isolation, car ce dernier ne correspond pas à la norme SQL. Pour pouvoir le mettre en œuvre il faut paramétrer la base de la sorte :

*NOTA : pour la démonstration, n'oubliez pas de reconstruire la base originale à l'aide du script n°1*

```
USE master
GO

ALTER DATABASE DB_ISO_LEVEL
SET ALLOW_SNAPSHOT_ISOLATION ON
GO
```

*Script n° 5*

Dès lors la nouvelle solution est la suivante :

```
--> fenêtre 1
USE DB_ISO_LEVEL
GO

SET TRANSACTION ISOLATION LEVEL SNAPSHOT
BEGIN TRANSACTION TRAN1

DECLARE @TOTAL INT

SELECT @TOTAL = SUM(COL)
FROM   T_ISO

WAITFOR DELAY '00:00:20'

SELECT @TOTAL = @TOTAL - SUM(COL)
FROM   T_ISO

SELECT @TOTAL AS TOTAL

COMMIT TRANSACTION
SET TRANSACTION ISOLATION LEVEL READ COMMITTED

USE master
GO
```

```
--> fenêtre 2
USE DB_ISO_LEVEL
GO

BEGIN TRANSACTION TRAN2

INSERT INTO T_ISO VALUES (4)

WAITFOR DELAY '00:00:20'

COMMIT TRANSACTION

USE master
GO
```

*Script n° 6*

L'inconvénient de cette méthode est de forcer des copies de pages de la base originale dans la *tempdb*. L'utilisation à outrance de ce niveau d'isolation peut donc engorger la base de données stockant les objets temporaires ce qui peut s'avérer pénalisant pour d'autres traitements.

## 4 - Conclusions

Le passage à un niveau d'isolation supérieur provoque la pose de verrous dont la granularité, ou la rétention augmente. Certes ces verrous empêchent les anomalies susvisées et c'est bien leur rôle, mais ils augmentent la contention du fait d'un blocage plus fort. Augmenter le niveau d'isolation accroît donc en proportion la durée de blocage mais aussi la probabilité d'obtenir des verrous mortel (*deadlocks* en anglais, ou encore interblocage, étreinte fatale...).
Ce serait donc une grave erreur de recourir systématiquement au niveau d'isolation le plus fort afin de s'affranchir de tout problème !
C'est d'ailleurs dans cet esprit que le mode d'isolation par défaut de toute session SQL Server est le mode READ COMMITTED, largement suffisant pour la plupart des traitements.

On le voit donc, le pilotage du niveau d'isolation est à apprécier au coup par coup en fonction du traitement effectué dans la transaction et du niveau de confiance que l'on veut sur les résultats. Par exemple accepter un niveau d'isolation READ UNCOMMITTED peut s'avérer payant lorsque l'on calcule des statistiques dont le résultat est censé indiquer une tendance et non la valeur exacte. En revanche, un tel niveau d'isolation ne doit jamais être utilisé lorsque des traitements portent sur des calculs comptables.

Bref, sans être affaire de spécialiste, le choix d'un niveau d'isolation est délicat et mérite d'être étudié de manière sereine.