Các Mức Isolation Level

Vũ Huy Tâm

Isolation level là một thuộc tính của transaction, qui định mức độ cô lập của dữ liệu mà transaction có thể truy nhập vào khi dữ liệu đó đang được cập nhật bởi một transaction khác. Khi một transaction cập nhật dữ liệu đang diễn ra, một phần dữ liệu sẽ bị thay đổi (ví dụ một số bản ghi của bảng được sửa đổi hoặc bị xóa bỏ, một số được thêm mới), vậy các transaction hoặc truy vấn khác xảy ra đồng thời và cùng tác động vào các bản ghi đó sẽ diễn ra thế nào? Chúng sẽ phải đợi đến khi transaction đầu hoàn thành hay có thể thực hiện song song, kết quả dữ liệu nhận được là trong khi hay sau khi cập nhật? Bạn có thể điều khiển những hành vi này thông qua việc đặt isolation level của từng transaction. SQL Server cung cấp các mức isolation level sau xếp theo thứ tự tăng dần của mức độ cô lập của dữ liệu: Read Uncommitted, Read Commited, Repeatable Read, và Serializable. Từ bản 2005 bắt đầu bổ sung thêm một loại mới là Snapshot. Phần còn lại của bài này sẽ đi vào chi tiết của từng loại.

1. Read Uncommitted

Khi transaction thực hiện ở mức này, các truy vấn vẫn có thể truy nhập vào các bản ghi đang được cập nhật bởi một transaction khác và nhận được dữ liệu tại thời điểm đó mặc dù dữ liệu đó chưa được commit (uncommited data). Nếu vì lý do nào đó transaction ban đầu rollback lại những cập nhật, dữ liệu sẽ trở lại giá trị cũ. Khi đó transaction thứ hai nhận được dữ liệu sai. Hãy tìm hiểu qua ví dụ sau:

```
CREATE TABLE dbo.Item (id INT, NAME VARCHAR(50))

INSERT INTO dbo.Item SELECT 1,'a'
INSERT INTO dbo.Item SELECT 2,'b'
INSERT INTO dbo.Item SELECT 3,'c'

SELECT * FROM dbo.Item
```

Nay bạn hãy mở hai cửa sổ trong Management Studio, ở cửa số thứ nhất bạn nhập vào:

```
BEGIN TRAN
UPDATE dbo.Item
  SET name = 'x'
  WHERE id>2
WAITFOR DELAY '00:00:10' --wait for 10 seconds
ROLLBACK
```

Và ở cửa số thứ hai bạn nhập:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
SELECT * FROM dbo.Item
```

Giờ bạn thực hiện đoạn lệnh ở cửa sổ thứ nhất rồi nhanh chóng chuyển sang thực hiện đoạn lệnh ở cửa sổ thứ hai. Bạn sẽ thấy cửa số thứ hai trả về bản ghi số 3 với name = ‘x’. Tuy nhiên sau đó transaction ở cửa số thứ nhất bị rollback và sau khi cả hai transaction kết thúc, bản ghi số 3 lại trở lại giá trị ban đầu name=’c’. Như vậy là transaction ở cửa số

thứ hai đã nhận được dữ liệu sai vì dữ liệu này chưa được commit. Hiện tượng này gọi là *uncommited read*, hay còn gọi là *dirty read*. Ưu điểm của mức isolation này là tăng độ tương tranh trong database, các tiến trình đọc không cần đợi đến khi tiến trình ghi hoàn tất mà có thể lấy dữ liệu ra được ngay. Nói nôm na là yêu cầu đọc của nó là "tôi không cần biết dữ liệu có đang được cập nhật hay không, hãy cho tôi dữ liệu hiện có ngay tại thời điểm này". Tùy theo ứng dụng của bạn mà bạn có thể đặt mức isolation này không, nếu việc đọc sai như trên là không thể chấp nhận được bạn cần đặt mức isolation cao hơn. Còn nếu có thể dung thứ được thì đặt mức này sẽ giúp tăng hiệu năng đọc cho hệ thống. Chú ý là mức isolation này tương được với gợi ý "NOLOCK" khi truy vấn bảng, đoạn lệnh ở cửa sổ thứ hai tương đương với:

```
SELECT * FROM dbo.Item WITH (NOLOCK)
```

2. Read Commited

Đây là mức isolation mặc định, nếu bạn không đặt gì cả thì transaction sẽ hoạt động ở mức này. Transaction sẽ không đọc được dữ liệu đang được cập nhật mà phải đợi đến khi việc cập nhật thực hiện xong. Vì thế nó tránh được dirty read như ở mức trên. Giờ hãy sửa lại đoạn lệnh ở cửa số thứ hai thành:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
SELECT * FROM dbo.Item
WHERE id>2
```

Và thực hiện lại hai cửa sổ theo trình tự như trên, bạn sẽ thấy cửa sổ thứ hai không trả về kết quả ngay mà phải đợi đến khi cửa số thứ nhất thực hiện xong. Và lần này cửa sổ thứ hai trả về dữ liệu đúng. Tuy nhiên nếu transaction thứ hai insert thêm bản ghi nằm trong phạm vi cập nhật của transaction thứ nhất, nó vẫn được phép làm như vậy và gây nhiễu đến transaction thứ nhất. Giờ hãy sửa lại code ở hai cửa sổ thành:
cửa số 1

```
BEGIN TRAN
UPDATE dbo.Item
SET name = 'x'
WHERE id>2
WAITFOR DELAY '00:00:10' --wait for 10 seconds
--ROLLBACK
COMMIT
SELECT * FROM dbo.Item
```

Cửa sổ hai:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
INSERT INTO dbo.Item SELECT 5,'e'
```

Sau khi thực hiện cả hai cửa sổ bạn sẽ thấy kết quả trả về có chứa bản ghi 5 với name = 'e'. Điều này hoàn toàn bất ngờ vì theo trình tự thực hiện đoạn lệnh ở cửa sổ thứ nhất, tất cả các bản ghi với id>2 đều được cập nhật. Trong tình huống trên, bản ghi 5 đã xuất hiện sau khi bảng được cập nhật nhưng trước khi transaction kết thúc. Vì thế nó được gọi là bản ghi ma (phantom row).

3. Repeatable read

Mức isolation này hoạt động như mức read commit nhưng nâng thêm một nấc nữa bằng cách ngăn không cho transaction ghi vào dữ liệu đang được đọc bởi một transaction khác cho đến khi transaction khác đó hoàn tất. Trở lại hai cửa sổ:
Cửa số 1:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRAN
SELECT * FROM dbo.Item
WAITFOR DELAY '00:00:10' --wait for 10 seconds
SELECT * FROM dbo.Item
COMMIT
```

Cửa sổ 2:

```
UPDATE dbo.Item
SET name = 'x'
WHERE id>2
SELECT * FROM item
```

Khi thực hiện code ở hai cửa sổ liên tiếp nhau, hai lệnh select ở cửa sổ 1 cho cùng kết quả và cửa sổ 2 phải đợi đến khi cửa sổ 1 hoàn tất mới được thực hiện. Mức isolation này đảm bảo các lệnh đọc trong cùng một transaction cho cùng kết quả, nói cách khác dữ liệu đang được đọc sẽ được bảo vệ khỏi cập nhật bởi các transaction khác. Tuy nhiên nó không bảo vệ được dữ liệu khỏi insert hoặc delete: nếu bạn thay lệnh update ở cửa sổ thứ hai bằng lệnh insert, hai lệnh select ở cửa sổ đầu sẽ cho kết quả khác nhau. Vì thế nó vẫn không tránh được hiện tượng bản ghi ma.

4. Serializable

Mức isolation này tăng thêm một cấp nữa và khóa toàn bộ dải các bản ghi có thể bị ảnh hưởng bởi một transaction khác, dù là UPDATE/DELETE bản ghi đã có hay INSERT bản ghi mới. Nếu bạn thay cửa sổ 1 bằng đoạn code

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRAN
SELECT * FROM dbo.Item
WAITFOR DELAY '00:00:10' --wait for 10 seconds
SELECT * FROM dbo.Item
COMMIT
```

và cửa sổ 2 bằng

```
INSERT INTO dbo.Item SELECT 4,'d'
```

Cửa sổ 2 sẽ bị treo đến khi cửa sổ 1 thực hiện xong, và hai lệnh SELECT trong cửa sổ 1 trả về kết quả giống nhau.

5. Snapshot

Mức độ này cũng đảm bảo độ cô lập tương đương với Serializable, nhưng nó hơi khác ở phương thức hoạt động. Khi transaction đang select các bản ghi, nó không khóa các bản ghi này lại mà tạo một bản sao (snapshot) và select trên đó. Vì vậy các transaction khác insert/update lên các bản ghi đó không gây ảnh hưởng đến transaction ban đầu. Tác dụng của nó là giảm blocking giữa các transaction mà vẫn đảm bảo tính toàn vẹn dữ liệu. Tuy nhiên cái giá kèm theo là cần thêm bộ nhớ để lưu bản sao của các bản ghi, và phần bộ nhớ này là cần cho mỗi transaction do đó có thể tăng lên rất lớn. Để thiết lập isolation mức này bạn cần đặt lại option của database:

```
ALTER DATABASE TestDB
SET ALLOW_SNAPSHOT_ISOLATION ON
```

Về phạm vi áp dụng các mức isolation

Các mức isolation từ 1 – 4 kể trên tăng theo thứ tự mức độ cô lập dữ liệu, giúp tăng tính toàn vẹn dữ liệu và nhất quán của transaction. Đồng thời nó cũng tăng thời gian chờ lẫn nhau của các transaction. Khi càng lên mức cao, đòi hỏi về tính toàn vẹn dữ liệu càng cao và càng có nhiều tình huống một transaction ngăn không cho các transaction khác truy nhập vào dữ liệu mà nó đang thao tác. Do đó nó càng tăng tình trạng locking và blocking trong database (ngoại trừ với snapshot thì tăng lượng bộ nhớ cần sử dụng). Hiệu năng của hệ thống do đó bị giảm đi. Thông thường, mức isolation read commited (mức mặc định) là phù hợp trong đa số các ứng dụng. Có thể một vài chức năng quan trọng (ví dụ chức năng ở trang admin update dữ liệu có ảnh hưởng đến toàn hệ thống) bạn cần tính toàn vẹn cao và phải chọn mức isolation cao hơn. Hoặc có những chức năng cần ưu tiên tốc độ thực hiện và có thể chấp nhận một chút dữ liệu không nhất quán, bạn có thể đặt xuống mức read uncommited. Bảng dưới đây tóm tắt các tính năng của từng mức isolation.

Mức Isolation	Dirty read	Nonrepeatable read	Phantom read
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable read	No	No	Yes
Serializable	No	No	No
Snapshot	No	No	No