



QuẢN LÝ GIAO DỊCH



GIAO DỊCH



KHÁI NIỆM

- Giao dịch: là một đơn vị thực hiện chương trình truy xuất vào dữ liệu và có thể làm thay đổi nội dung của nhiều hạng mục dữ liệu
- Là một tập các lệnh được thực hiện như một khối lệnh, có thể thành công hoàn toàn hoặc thất bại hoàn toàn

4 TÍNH CHẤT CỦA GIAO DỊCH

- ❖ Atomic (nguyên tử): đảm bảo toàn bộ hoạt động của giao dịch thành công hoặc thất bại.
- ❖ Consistency (tính nhất quán): khi transaction hoàn thành, dữ liệu phải ở trạng thái toàn vẹn
- ❖ Isolation (cô lập): khi có nhiều giao dịch thực hiện đồng thời thì phải đảm bảo chúng được giữ độc lập để các kết quả không ảnh hưởng lẫn nhau
- ❖ Durability (bền vững): sau khi giao dịch thực hiện, nếu có sự cố thì tất cả các dữ liệu thay đổi trong giao dịch vẫn được khôi phục lại theo yêu cầu của giao dịch

VÍ DỤ

- Một nhà băng gồm các tài khoản. Một giao dịch T chuyển 50 từ tài khoản A sang tài khoản B. Giả sử tài khoản A và B tương ứng là 1000 và 2000
- Giao dịch này có thể được viết như sau:

READ(A)

A=A-50

WRITE(A)

READ(B)

B=B+50

WRITE(B)

Ví dụ

❖ Tính Atomic:

- Nếu như giao dịch thành công thì tất cả các lệnh trong giao dịch thành công và $A=950, B=2050$
- Giao dịch đang được thực hiện đến lệnh $READ(B)$ mà hệ thống xảy ra sự cố thì toàn bộ giao dịch bị hủy: toàn bộ các câu lệnh trong giao dịch đều không được thực hiện thành công $\Rightarrow A=1000$ và $B=2000$



Ví dụ

- Tính Consistency:
- Sau khi transaction hoàn thành, dữ liệu phải ở trạng thái nhất quán: tài khoản A phải có số tiền là 950 và B có 2050.

Ví dụ

- Durability: sau khi giao dịch thực hiện thành công, giả sử có sự cố thì dữ liệu sau khi khôi phục phải đảm bảo là A có 950 và B có 2050.
- Isolation: Nếu như giao dịch chuyển tiền đang thực hiện đến câu lệnh READ(A) thì có 1 giao dịch khác cũng thực hiện việc chuyển 1000 từ A sang C. Khi đó hai giao dịch sẽ tương tranh với nhau

Tại sao cần phải quản lý giao dịch?

- Giả sử có 2 transaction truy xuất đồng thời trên 1 đơn vị dữ liệu.

T1	T2	Nhận xét
Đọc	Đọc	Không có tranh chấp
Đọc	Ghi	Xảy ra tranh chấp
Ghi	Đọc	Xảy ra tranh chấp
Ghi	Ghi	Xảy ra tranh chấp

- Để giải quyết các tranh chấp => phải quản lý các giao dịch (sử dụng các mức cô lập hoặc khóa) để khi có tranh chấp xảy ra thì sẽ xác định xem transaction nào được ưu tiên, transaction nào phải chờ



Tại sao cần phải có các mức cô lập/khóa

- Để hạn chế quyền truy cập trong môi trường đa người dùng
- Để đảm bảo tính toàn vẹn của CSDL: dữ liệu bên trong CSDL có thể bị sai về logic, các query chạy trên đó sẽ đưa ra các kết quả không như mong đợi
- Khi một giao dịch muốn truy cập riêng vào một bảng, server sẽ khóa/cô lập bảng đó lại cho riêng giao dịch đó



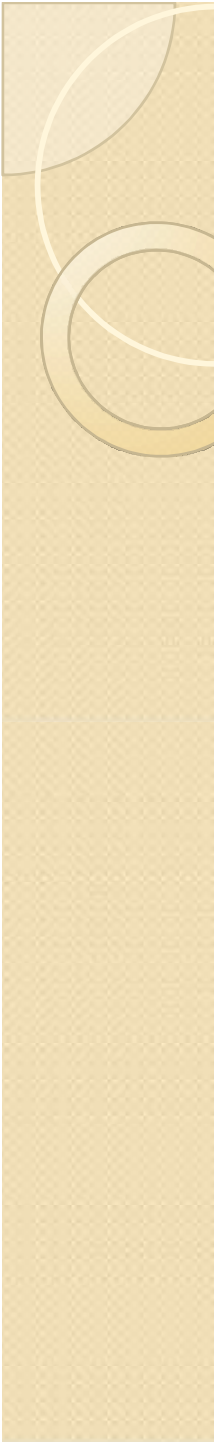
Những vấn đề trong truy xuất đồng thời

- Dirty reads (đọc dữ liệu sai)
- Lost updates (mất dữ liệu cập nhật)
- Unrepeatable reads (không thể đọc lại)
- Phantoms (đọc bản ghi không có)



Những vấn đề trong truy xuất đồng thời

- Dirty reads (đọc dữ liệu sai): xảy ra khi giao dịch đọc bản ghi mà bản ghi đó đang được truy xuất bởi 1 giao dịch khác chưa hoàn thành.
- Do dữ liệu chưa được commit và chúng ta đọc phải dữ liệu cũ, dữ liệu sai.

- 
- Ví dụ: ◦ Tài khoản A có 8 tỷ
 - Vào thời điểm t_1 , giao dịch T1 chuyển 2 tỷ từ A sang B
 - Vào thời điểm $t_2 \geq t_1$, giao dịch T2 cũng thực hiện chuyển 7 tỷ từ A sang C
 - Nếu tại thời điểm t_2 , giao dịch T1 chưa hoàn thành, giao dịch T2 sẽ đọc thấy trong tài khoản A vẫn còn 8 tỷ \Rightarrow Tính nhất quán bị phá vỡ, ngân hàng bị mất tiền

Những vấn đề trong truy xuất đồng thời

- Lost update: xảy ra khi nhiều giao dịch cùng lúc muốn cập nhật 1 đơn vị dữ liệu. Khi đó, tác dụng của giao dịch cập nhật sau sẽ ghi đè lên tác dụng cập nhật của giao dịch trước
- Ví dụ: Hệ thống bán vé máy bay online còn 500 vé
 - Vào lúc t1, giao dịch A bán 100 vé, đọc dữ liệu thấy còn 500 vé
 - Vào lúc t1, giao dịch B bán 200 vé, đọc dữ liệu thấy còn 500 vé
 - Vào lúc t2, giao dịch A cập nhật số vé còn lại: $500 - 100 = 400$
 - Vào lúc t3, giao dịch B cập nhật số vé còn lại: $500 - 200 = 300$.
 - Thao tác cập nhật của A bị mất .

Những vấn đề trong truy xuất đồng thời

- Unrepeatable reads: xảy ra khi giao dịch đọc một bản ghi 2 lần mà lần đọc sau cho kết quả khác lần đọc trước
- Ví dụ: ban đầu công ty có 100 nhân viên lương 5 triệu
 - Vào lúc t1, giao dịch A phải đếm xem có bao nhiêu nhân viên lương 5 triệu=>100
 - Vào lúc t2, giao dịch B cập nhật lương nhân viên Tâm thành 6 triệu (Tâm ban đầu lương là 5 triệu)
 - Vào lúc t3, giao dịch A lại phải đếm xem có bao nhiêu nhân viên lương 5 triệu=>99
 - Giá trị mà A đọc được lúc t1, t3 là mâu thuẫn nhau.

Những vấn đề trong truy xuất đồng thời

- Phantoms (bản ghi ma): xảy ra khi giao dịch đọc những bản ghi mà nó không mong muốn
- Ví dụ: giao dịch A cần tổng hợp 5 bản ghi 1,2,3,4,5 để làm một bản báo cáo
- t1: A đọc và đưa các bản ghi 1,2,3,4 vào báo cáo
- t2: giao dịch B xóa bản ghi 5 và thay bằng bản ghi 6
- t3: A đọc tiếp và đưa bản ghi 6 vào báo cáo
- Vậy báo cáo này vừa bị thiếu dữ liệu, vừa bị thừa dữ liệu.



Các mức độ cô lập

- Read Uncommitted
- Read Committed
- Repeatable Read
- Serializable

Read Uncommitted

- Giao tác đọc dữ liệu mà không cần quan tâm dữ liệu đó có đang bị thay đổi bởi giao tác khác không.
 - Ưu điểm: tăng hiệu năng đọc của các tiến trình
 - Nhược điểm: không ngăn chặn được 4 vấn đề trong tương tranh
- => Tùy vào ứng dụng để đặt mức isolation. Nếu việc đọc sai có thể chấp nhận được thì không cần đặt mức isolation cao hơn để tăng hiệu năng đọc cho hệ thống.

Ví dụ

- Read Uncommitted: Bảng test có dữ liệu như sau

ID	Name
1	a
2	b
3	C

T1	T2
<pre>begin tran update test set Name = 'x' where ID=3 waitfor delay '00:00:10' Rollback tran</pre>	<pre>begin tran set tran isolation level read uncommitted select * from test commit tran</pre>

- Kết quả: T2 nhận được gì?

ID	Name
1	a
2	b
3	d

- Giải thích vì sao? T2 đọc phải dữ liệu bản là bản ghi thứ 3



Read Committed

- Khi transaction được đặt ở mức độ cô lập này, nó không được phép đọc dữ liệu đang cập nhật mà phải đợi đến khi giao dịch đó hoàn tất

Read Committed

ID	Name
1	a
2	b
3	c

T1	T2
<pre>begin tran update test set Name ='x' where id=3 waitfor delay '00:00:10' Rollback tran</pre>	<pre>begin tran set tran isolation level read committed select * from test commit tran</pre>

Read Committed

- Kết quả: T2 nhận được gì?

ID	Name
1	a
2	b
3	c

- Giải thích vì sao?
- Mức Read Committed ngăn không cho phép giao dịch đọc (select/delete/update) CSDL khi giao dịch khác đang thay đổi (insert/delete/update) CSDL đó



Read Committed

- Ngăn được Dirty Reads, Lost Update
- Không ngăn được hiện tượng
Unrepeatable Read, Phantom

Repeatable Read

- Ngăn không cho transaction cập nhật vào dữ liệu đang được đọc bởi transaction khác cho đến khi transaction đó hoàn tất việc đọc.
- Ưu điểm: giải quyết được vấn đề lost update, unrepeatable read
- Nhược điểm: chưa giải quyết được vấn đề phantom

Repeatable Read

- Bảng dữ liệu:

ID	Name
1	a
2	b
3	c

T1		T2	
begin tran set tran isolation level read committed select * from test waitfor delay '00:00:10' select * from test commit tran		begin tran set tran isolation level read committed Update test set name='x' where id=3 commit tran	
Id	name	id	name
1	a	1	a
2	b	2	b
3	c	3	x

Repeatable Read

T1		T2	
BEGIN TRAN set tran isolation level repeatable read SELECT * FROM test Wait for delay '00:00:10' SELECT * FROM test Commit tran		begin tran Update test set name='x' where id=3 commit tran	
id	name	id	name
1	a	1	a
2	b	2	b
3	c	3	x

Serializable

- Mức Repeatable bảo vệ được dữ liệu khỏi câu lệnh UPDATE nhưng không bảo vệ được khỏi câu lệnh INSERT và DELETE
- Mức Serializable bắt buộc các giao tác khác phải chờ đợi cho đến khi giao tác đó hoàn thành nếu muốn thay đổi dữ liệu
- Ưu điểm: giải quyết được vấn đề phantom
- Nhược điểm: làm chậm hoạt động của các giao dịch trong hệ thống

Serializable

A	B
a	1
b	2
c	3

T1				T2			
BEGIN TRAN set tran isolation level repeatable read SELECT * FROM test Wait for delay '00:00:10' SELECT * FROM test Commit tran				begin tran Insert into test values ('d', 5) Select * from test commit tran			
A	B	A	B	A	B	A	B
a	1	a	1	a	1	a	1
b	2	b	2	b	2	b	2
c	3	c	3	c	3	c	3
		d	5	d	5	d	5

Serializable

T1				T2	
BEGIN TRAN set tran isolation level serializable SELECT * FROM test Wait for delay '00:00:10' SELECT * FROM test Commit tran				begin tran Insert into test values ('d', 5) Select * from test commit tran	
A	B	A	B	A	B
a	1	a	1	a	1
b	2	b	2	b	2
c	3	c	3	c	3
				d	5



CÁC LOẠI GIAO DỊCH

- Giao dịch tường minh
- Giao dịch ngầm định
- Giao dịch tự động



GIAO DỊCH TỰỜNG MINH (EXPLICIT TRANSACTION)

- Là giao dịch phải khai báo trước
- BEGIN TRAN: giao dịch bắt đầu
- COMMIT TRAN: giao dịch thành công
- ROLLBACK TRAN: quay trở về trạng thái tại thời điểm ban đầu hay về một điểm dừng nào đó trong giao dịch

GIAO DỊCH TỰỜNG MINH (EXPLICIT TRANSACTION)

- Ví dụ: Nhân viên (MaNV, HoTen, MaPhong)
Phong (Map, TenP, Soluong)
- Viết một giao dịch để thay đổi phòng làm việc của một nhân viên 'NV01' sang phòng 'NS'

BEGIN TRAN

UPDATE NHANVIEN SET MaPHONG='NS'
WHERE MaNV='NV01';

COMMIT TRAN

=> Giao dịch trên thiếu tính chất gì?

GIAO DỊCH TỰỜNG MINH (EXPLICIT TRANSACTION)

BEGIN TRAN

Declare @old_dept char(10);

Select @old_dept=Maphong from NHANVIEN
where MaNV='NV01';

UPDATE NHANVIEN SET PHONG='NS'
WHERE MaNV='NV01';

UPDATE Phong SET soluong=soluong+1 WHERE
MaPhong='NS';

UPDATE Phong SET soluong=soluong-1
WHERE MaPhong=@old_dept;

COMMIT TRAN

GIAO DỊCH TỰỜNG MINH (EXPLICIT TRANSACTION)

- Ví dụ: Bảng TAIKHOAN cho biết số tài khoản (SOTK) và số tiền trong tài khoản (SOTIEN).Viết giao dịch để chuyển 100 từ số tài khoản A sang số tài khoản B

BEGIN TRAN

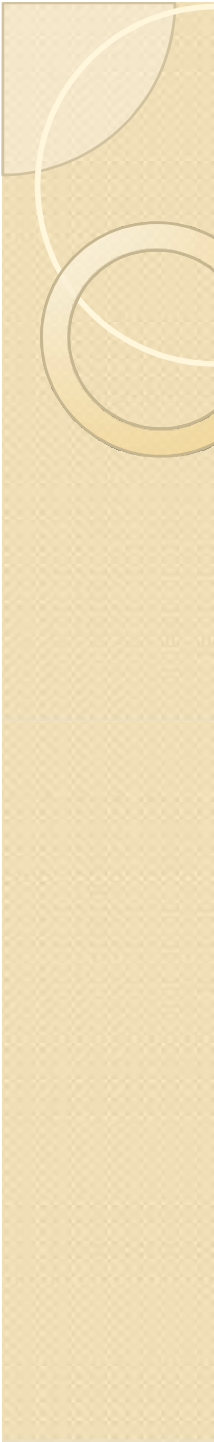
UPDATE TAIKHOAN set SOTIEN=SOTIEN-100
where SOTK='A'

UPDATE TAIKHOAN set SOTIEN=SOTIEN+100
where SOTK='B'

Else Begin

COMMIT TRAN

END



```
BEGIN TRAN
IF (SELECT SOTIEN FROM TAIKHOAN WHERE SOTK='A') < 100
BEGIN
PRINT N'Số tiền trong tài khoản không cho phép bạn thực hiện
giao dịch'
ROLLBACK TRAN
END
ELSE
BEGIN
UPDATE TAIKHOAN set SOTIEN=SOTIEN-100 where SOTK='A'
UPDATE TAIKHOAN set SOTIEN=SOTIEN+100 where SOTK='B'
COMMIT TRAN
END
```

GIAO DỊCH KHÔNG TỰỜNG MINH (IMPLICIT TRANSACTION)

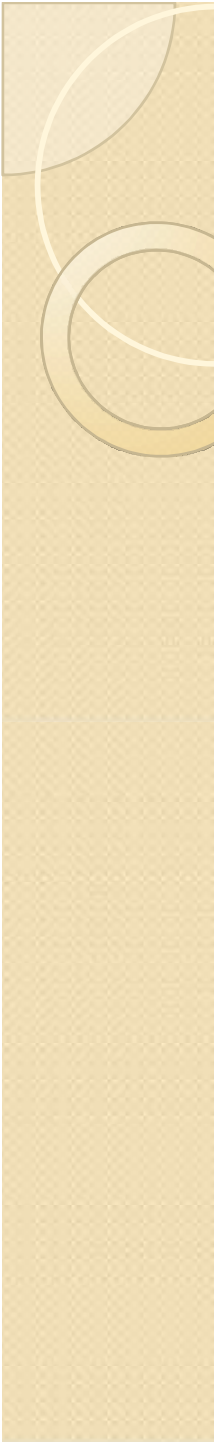
- Là giao dịch ngầm định. Nó không yêu cầu phát biểu BEGIN TRAN. Bản thân nó được tự động khởi tạo.
- Trong SQL Server, implicit transaction mặc định ở chế độ nghỉ
- Bật chế độ làm việc:
`SET implicit_transactions ON`
- Tắt chế độ làm việc:
`SET implicit_transactions OFF`

GIAO DỊCH KHÔNG TỰ ĐỘNG MINH

- Sau khi chế độ Transaction implicit đã được bật ON cho một kết nối, SQL Server tự động bắt đầu một transaction khi nó thực thi bất kỳ các lệnh sau:
 - ALTER TABLE
 - REVOKE
 - CREATE
 - SELECT
 - DELETE
 - INSERT
 - UPDATE
 - DROP
 - OPEN
 - FETCH
 - TRUNCATE TABLE
 - GRANT

AUTOCOMMIT TRANSACTION

- Chế độ chuyển giao tự động là chế độ mặc định quản lý các transaction của SQL SERVER
- Một lệnh sẽ được tự động committed nếu nó thực hiện thành công hoặc sẽ rollback nếu nó gặp lỗi

- 
- Khi gặp thông báo BEGIN TRAN, SQL Server chuyển từ chế độ autocommit sang chế độ explicit
 - SQL Server chuyển về chế độ autocommit khi transaction tựờng mình đã được chuyển giao (commit) hay trả ngược về đầu (roll back) hay khi mode transaction ngầm định bị tắt.

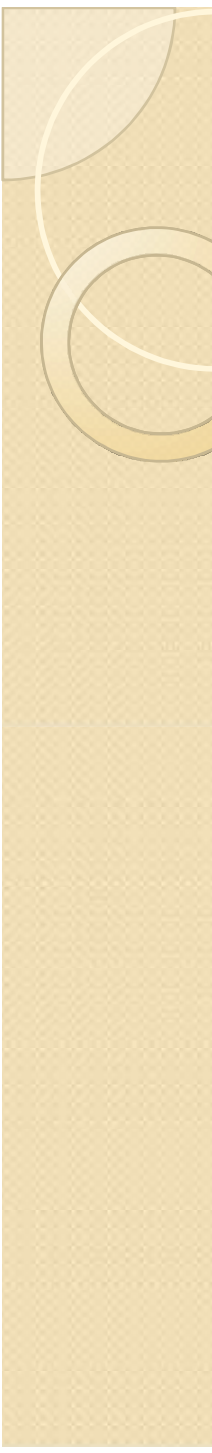
LOCKS

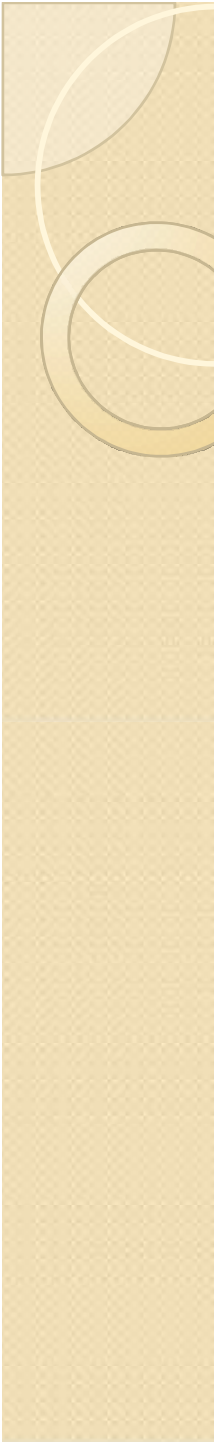
- Khóa (Lock) để giới hạn quyền truy nhập trong môi trường đa người dùng
- SQL Server sử dụng LOCK để đảm bảo tính toàn vẹn của giao dịch và tính thống nhất của database
- Nếu không sử dụng lock, dữ liệu bên trong CSDL có thể sai về mặt logic. Các query chạy trên đó sẽ đưa ra kết quả không như mong đợi
- Bản chất của lock là việc người dùng muốn truy cập vào riêng một bảng, server sẽ lock bảng đó lại cho riêng người đó.

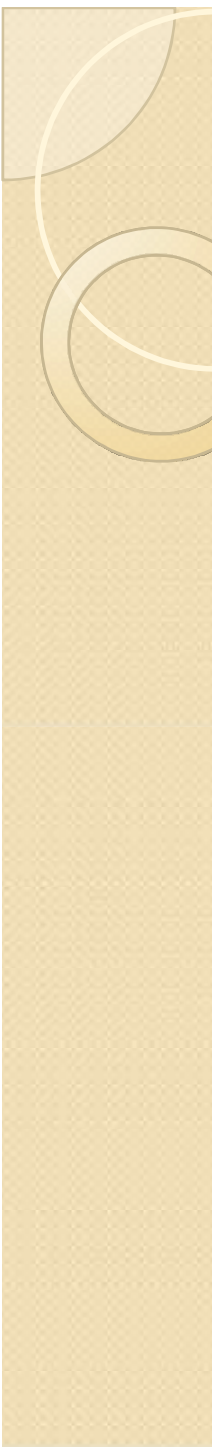


Các loại LOCK trong SQL Server

- Shared Lock (Khóa chia sẻ)
- Exclusive Lock (Khóa độc quyền)
- Update Lock (Khóa cập nhật)

- 
- Shared Lock: khi 1 đơn vị dữ liệu được thiết lập Shared Lock bởi giao dịch A thì:
 - Các giao dịch khác được đọc dữ liệu nhưng không được thay đổi dữ liệu.
 - Tại một thời điểm có thể có nhiều Shared Lock trên cùng một đơn vị dữ liệu.
 - Đơn vị dữ liệu có thể là một dòng, một bảng, một trang

- 
- Exclusive Lock: khi 1 đơn vị dữ liệu được thiết lập Exclusive Lock bởi giao dịch A thì các giao dịch khác không được truy cập vào đơn vị dữ liệu đó.
 - Tại một thời điểm, chỉ có tối đa một giao tác có khóa exclusive lock trên 1 đơn vị dữ liệu.
 - Khi một đơn vị đang có Shared Lock thì có thể thiết lập Exclusive Lock trên đơn vị dữ liệu đó không?
 - Khi một đơn vị đang có Exclusive Lock thì có thể thiết lập một Shared Lock lên đơn vị dữ liệu đó không?

- 
- Update Lock:
 - Là chế độ khóa trung gian giữa Shared Lock và Exclusive Lock
 - Cho phép đọc dữ liệu và ghi lại dữ liệu sau khi đọc dữ liệu này

Chỉ định khóa trong từng lệnh

- Khi đặt các mức isolation level, mức cô lập được chỉ định sẽ tác dụng lên toàn bộ câu lệnh nằm ngay sau nó.
- Nếu một lệnh không được chỉ định lock trực tiếp, nó sẽ hoạt động theo mức cô lập chung hiện hành của transaction

Chỉ định khóa trong từng lệnh

- Cú pháp :

SELECT ...FROM TABLE WITH (LOCK)

DELETE... FROM TABLE WITH (LOCK)

Ví dụ: Select * from test with (nolock)

Chỉ định khóa trong từng lệnh

- Một số lock trong SQL Server

READUNCOMMITTED/NOLOCK	Tương tự như mức READ UNCOMMITTED
READCOMMITTED	Tương tự như mức READ COMMITTED
REPEATABLE READ	Tương tự như mức REPEATABLE READ
SERIALIZABLE/HOLDLOCK	Tương tự như mức SERIALIZABLE
XLOCK	Khóa độc quyền
UPDLOCK	Khóa update
ROWLOCK	Khóa chỉ những dòng thao tác
TABLOCK	Khóa bảng
TABLOCKX	Xlock+tablock