

Chương 7

Quản lý giao tác & Khóa

Nội dung

1. Định nghĩa giao tác (Transactions)
2. Thuộc tính của giao tác
3. Các loại giao tác
4. Công dụng Transaction log
5. Định nghĩa khóa (Lock)
6. Các vấn đề đồng thời (Concurrency problem).
7. Các kiểu khóa

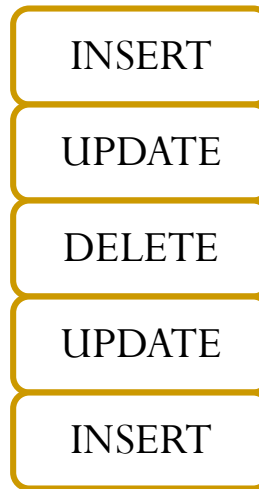
Transaction

- Một giao tác (transaction) là một chuỗi gồm một hoặc nhiều câu lệnh SQL được kết hợp lại với nhau thành một khối công việc.
- Các câu lệnh SQL xuất hiện trong giao tác thường có mối quan hệ tương đối mật thiết với nhau và thực hiện các thao tác độc lập. Việc kết hợp các câu lệnh lại với nhau trong một giao tác nhằm đảm bảo tính toàn vẹn dữ liệu và khả năng phục hồi dữ liệu.
- Trong một giao tác, các câu lệnh có thể độc lập với nhau nhưng tất cả các câu lệnh trong một giao tác đòi hỏi hoặc phải thực thi trọn vẹn hoặc không một câu lệnh nào được thực thi.

Transaction

VD: giao tác chuyển khoản 50\$ từ tài khoản A sang tài khoản B

read(A)
 $A := A - 50$
write(A)
read(B)
 $B := B + 50$
write(B)

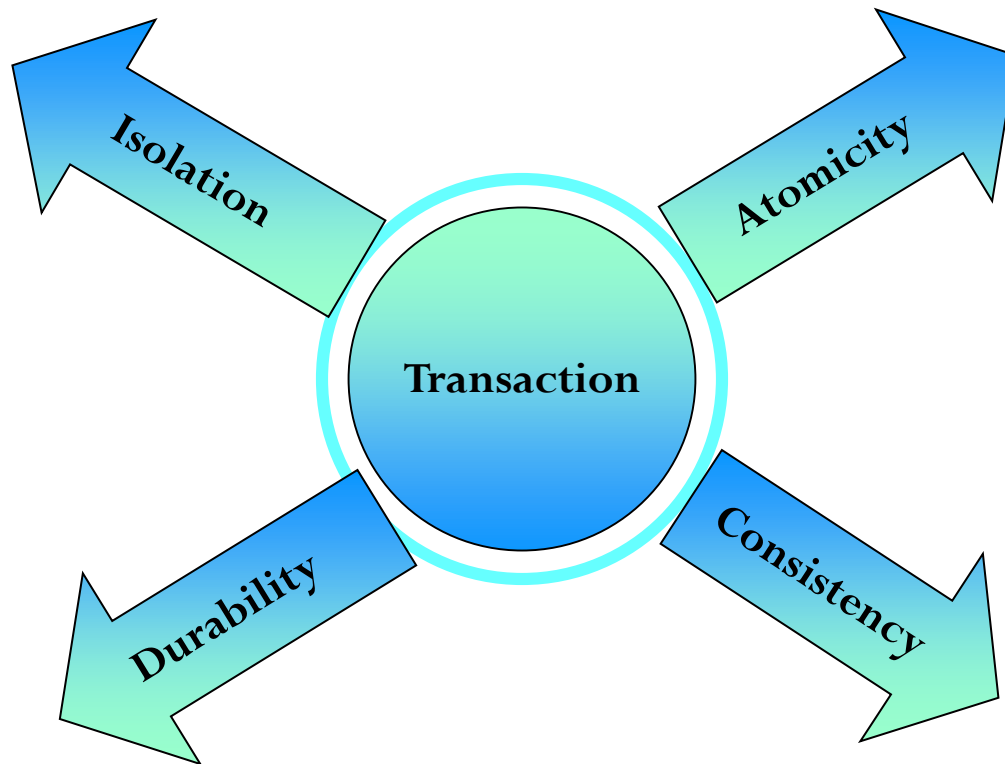


Transaction

Thuộc tính của Transaction (**ACID**)

- ❑ **Tính nguyên tử (Atomicity)**: Là một công việc nguyên tố. Hoặc toàn bộ các hiệu chỉnh dữ liệu được thực hiện hoặc là tất cả chúng đều không được thực hiện.
- ❑ **Tính nhất quán (Consistency)**: Tính nhất quán đòi hỏi sau khi giao tác kết thúc, cho dù là thành công hay bị lỗi, tất cả dữ liệu phải ở trạng thái nhất quán (tức là sự toàn vẹn dữ liệu phải luôn được bảo toàn)
- ❑ **Tính cô lập (Isolation)**: Dữ liệu khi hiệu chỉnh được thực hiện bởi các transaction phải độc lập với các hiệu chỉnh khác của các transaction đồng thời khác.
- ❑ **Tính bền vững (Durability)**: Sau khi một giao tác thực hiện thành công, các thay đổi đã được tạo ra đối với CSDL vẫn còn ngay cả khi xảy ra sự cố hệ thống.

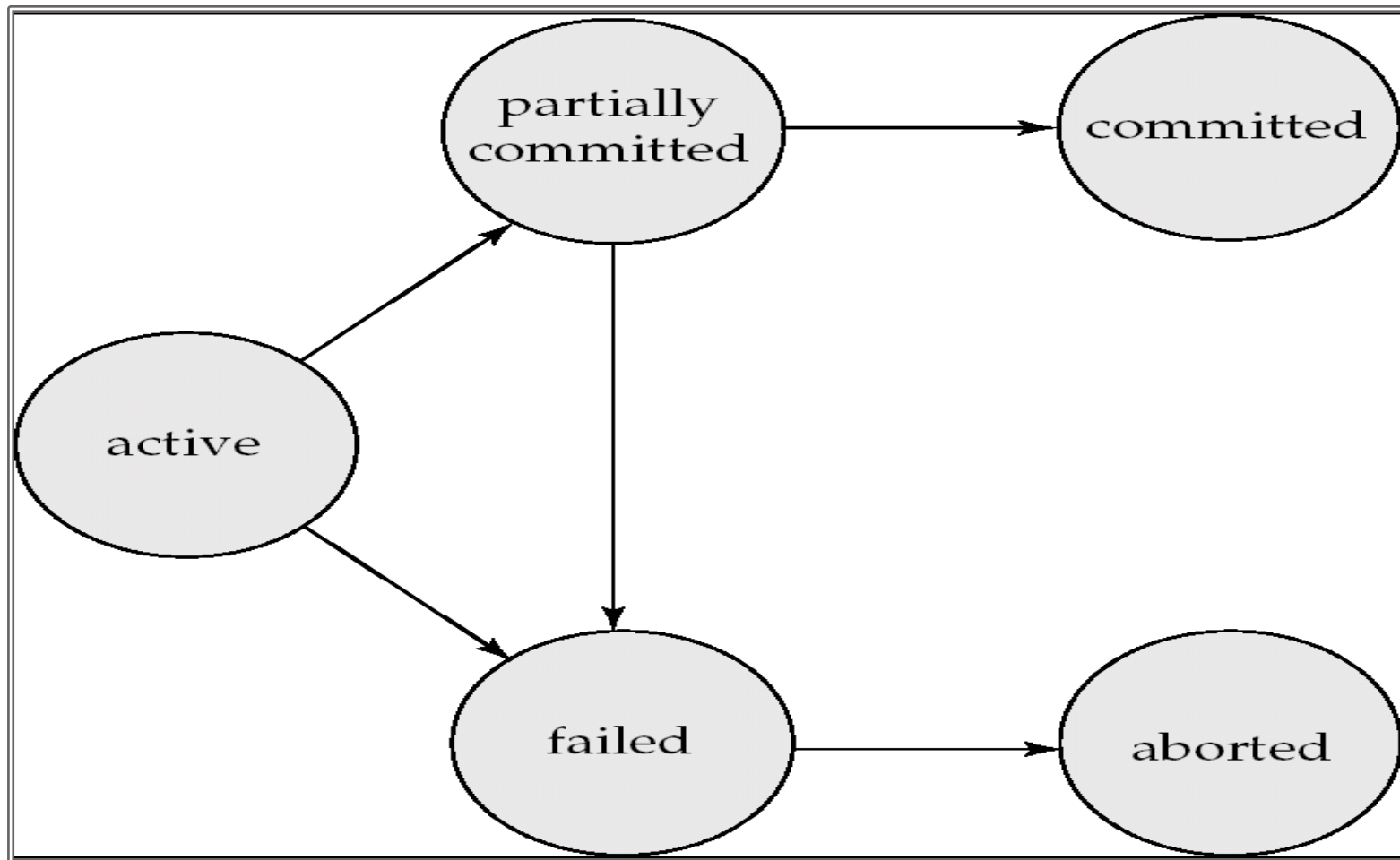
Transaction Properties



Các trạng thái của transaction

- ❑ **Active (kích hoạt)** : Trạng thái khởi đầu, giao tác giữ trong trạng thái này trong khi nó đang thực hiện
- ❑ **Partially committed (chuyển giao 1 phần)**: Sau khi lệnh cuối cùng được thực hiện.
- ❑ **Failed (thất bại)**: Sau khi phát hiện rằng sự thực hiện không thể tiếp tục được nữa.
- ❑ **Aborted (hủy)**: Sau khi giao tác đã bị “roll back” (quay lui) và CSDL đã phục hồi lại trạng thái của nó trước khi khởi động giao tác.
- ❑ **Committed (đã chuyển giao)**: Sau khi thực hiện thành công giao tác.

Các trạng thái của transaction



Mô hình Transaction trong SQL

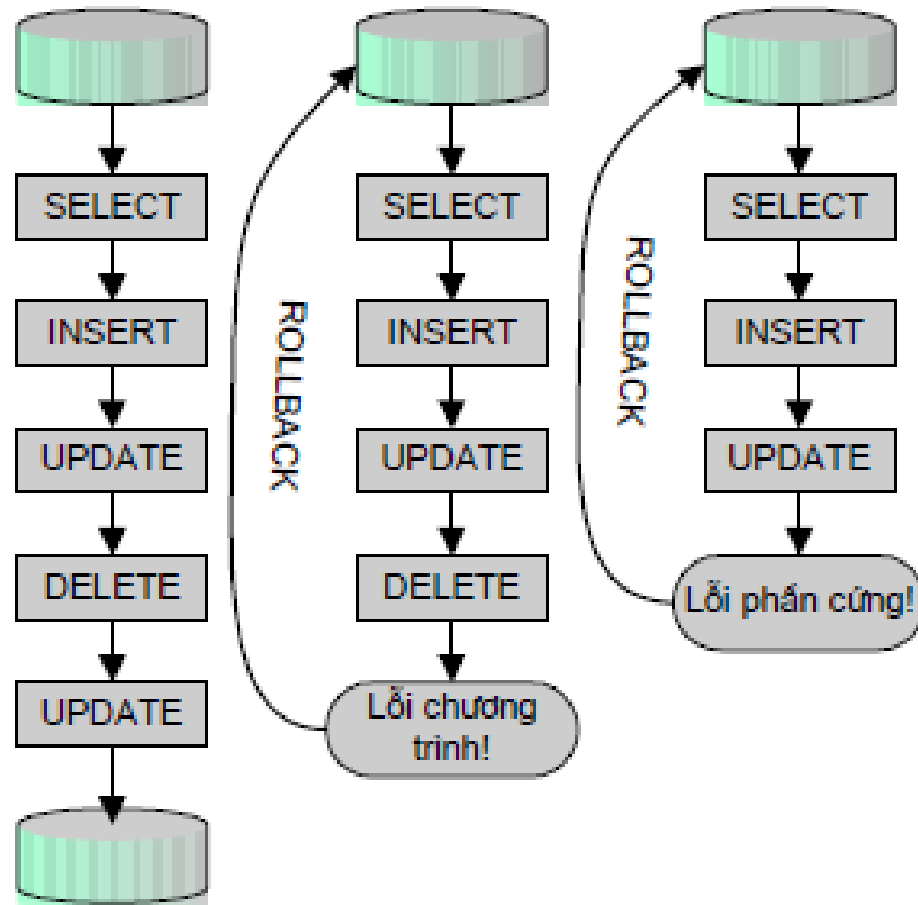
- ❑ Giao tác SQL được định nghĩa dựa trên các câu lệnh xử lý giao tác sau đây:
 - BEGIN TRANSACTION: Bắt đầu một giao tác
 - SAVE TRANSACTION: Đánh dấu một vị trí trong giao tác (gọi là điểm đánh dấu).
 - ROLLBACK TRANSACTION: Quay lui trở lại đầu giao tác hoặc một điểm đánh dấu trước đó trong giao tác.
 - COMMIT TRANSACTION: Đánh dấu điểm kết thúc một giao tác. Khi câu lệnh này thực thi cũng có nghĩa là giao tác đã thực hiện thành công.
 - ROLLBACK [WORK]: Quay lui trở lại đầu giao tác.
 - COMMIT [WORK]: Đánh dấu kết thúc giao tác.

Các trạng thái của transaction

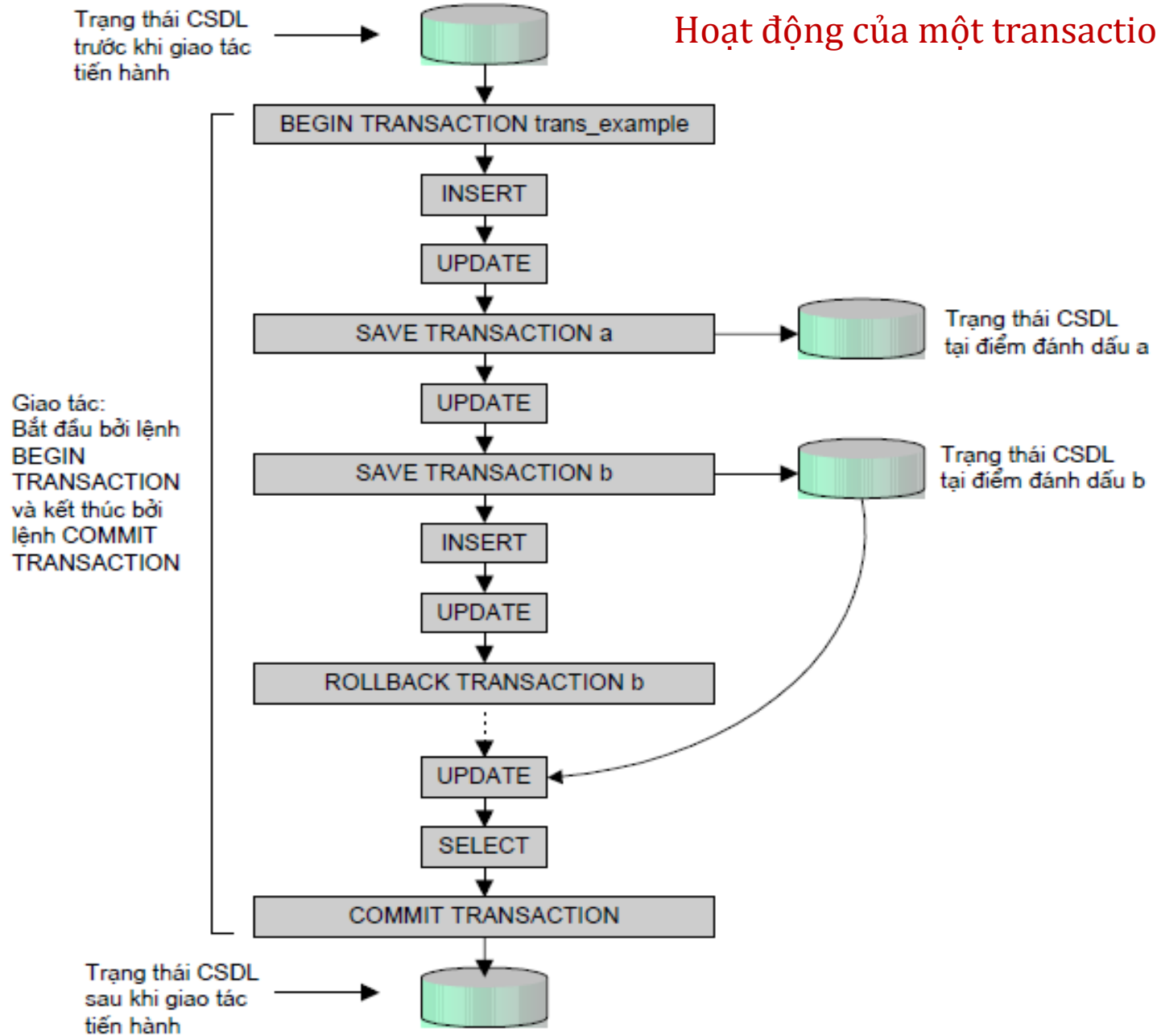
Trạng thái CSDL
trước khi giao tác
tiến hành

Giao tác

Trạng thái CSDL
sau khi giao tác
tiến hành



Hoạt động của một transaction



Các loại Transaction

- Explicit – Tường minh
- Implicit – Không tường minh
- Auto commit transaction - Giao tác tự động chuyển giao
- Distributed Transactions

Các loại Transaction

❑ Tường minh - Explicit

- Được khai báo bắt đầu bằng lệnh BEGIN TRAN
- ROLL BACK: tất cả sự thay đổi bị từ bỏ. CSDL “roll back” về trạng thái nhất quán trước đó
- COMMIT: hoàn thành giao tác, tất cả sự thay đổi được phản ánh trong CSDL

❑ Không tường minh - Implicit

- Giao tác mới sẽ tự động bắt đầu ngay khi giao tác trước đó hoàn tất
- Giao tác “roll back” khi chương trình kết thúc bất thường.
- Giao tác kết thúc khi chương trình kết thúc

Các loại Transaction

- ❑ Giao tác tự động chuyển giao (auto commit transaction): mỗi một lệnh được xem như một transaction
- ❑ Distributed Transactions: Là một loại Explicit Transaction nhưng liên quan đến nhiều Server. Sự quản lý phải được kết hợp giữa các bộ quản lý tài nguyên của các server và do bộ quản lý giao tác (transaction manager) làm.

Transaction log

- ❑ Dùng để theo vết tất cả các giao tác
- ❑ Phục hồi dữ liệu
- ❑ Một transaction log gồm:
 - Một record đánh dấu bắt đầu 1 transaction
 - Thông tin về transaction
 - Thao tác (cập nhật, xóa, chèn)
 - Tên các object (thành phần dữ liệu) ảnh hưởng bởi transaction
 - Giá trị trước và sau của các field được cập nhật.
 - Con trỏ trỏ đến dòng trước và sau trong cùng 1 transaction
 - Một record đánh dấu kết thúc transaction

Transaction Log

TABLE 9.1 A TRANSACTION LOG

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	1558-QW1	PROD_QOH	25	23
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	525.75	615.73
365	101	363	Null	COMMIT	**** End of Transaction				



TRL_ID = Transaction log record ID

PTR = Pointer to a transaction log record ID

TRX_NUM = Transaction number

(Note: The transaction number is automatically assigned by the DBMS.)

Giao tác tường minh

- Transaction phải được định nghĩa: bắt đầu bằng Begin Transaction và kết thúc bằng Commit Transaction
- Dùng BEGIN TRANSACTION để bắt đầu
BEGIN TRAN [SACTION]
(Transaction_Name|@Tran_Name_Variable)
- Dùng COMMIT TRANSACTION hay COMMIT WORK để kết thúc giao tác 1 cách mỹ mãn
COMMIT [TRAN [SACTION]
(Transaction_Name|@Tran_Name_Variable)

Giao tác tường minh

- Lưu giao tác

SAVE TRAN [SACTION]

{Transaction_Name|@Tran_Name_Variable}

- Dùng ROLLBACK TRANSACTION hay ROLLBACK WORK để quay lui một transaction

ROLLBACK TRAN [SACTION]

[Transaction_Name|@Tran_Name_Variable

| savepoint_Name |@savepoint variable]

- @@TRANCOUNT : Trả về số thứ tự Transaction được mở, tối đa lồng 32 cấp, không nên lồng nhau.

Giao tác ngầm định

- Khi một Connection đang mở trong chế độ Implicit, SQL Server bắt đầu một transaction mới một cách tự động sau khi transaction hiện hành hoàn tất hoặc roll back (quay lui).
- Bạn không cần bắt đầu một transaction, bạn chỉ cần Commit hay Rollback mỗi transaction.
- Chế độ Implicit transaction (giao tác chạy ngầm) phát sinh một chuỗi các transaction liên tục.

Giao tác ngầm định

- 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**

Distributed Transactions

- Là một loại Explicit Transaction nhưng giao tác của nó liên quan nhiều Server. Sự quản lý phải được kết hợp giữa các bộ quản lý tài nguyên của các server và điều này gọi là transaction manager.
- Các Transaction trong một server là những tham chiếu từ nhiều Database, thực ra cũng là một Distributed Transaction.
- Transaction log: dùng để ngăn chặn người dùng hiệu chỉnh dữ liệu ảnh hưởng từ các transaction chưa hoàn tất.

Công dụng của Transaction

- ❖ Phục hồi các Transaction đặc biệt: Khi một ứng dụng ra lệnh ROLL BACK hay SQL nhận ra một lỗi, thì bảng ghi log được dùng để Roll Back bất kỳ hiệu chỉnh nào trong suốt quá trình Transaction chưa hoàn tất.
- ❖ Phục hồi tất cả các Transaction chưa hoàn tất khi SQL Server được bắt đầu.
- ❖ Hoàn trả lại Database lại đến một thời điểm bị lỗi nhằm đảm bảo không phát sinh mâu thuẫn khi có sự cố.

Chuyển giao tự động các Transaction

- ❑ Autocommit Transaction: Mô hình chuyển giao tự động là mô hình quản lý transaction mặc định của SQL Server.
- ❑ Một lệnh (statement) được chuyển giao (committed) nếu nó thực hiện thành công hay sẽ quay lui lại vị trí ban đầu (roll back) nếu nó gặp lỗi.

Chuyển giao tự động các Transaction

- ❑ Lệnh BEGIN TRANSACTION vượt quyền chế độ tự động chuyển giao (autocommit) mặc định.
- ❑ SQL Server trở về chế độ autocommit khi transaction tường minh đã được chuyển giao (commit) hay trả ngược về đầu (roll back) khi chế độ transaction ngầm định bị tắt.

Các lệnh không hợp lệ trong Transactions

- ALTER DATABASE
- DROP DATABASE
- RESTORE DATABASE
- CREATE DATABASE
- DISK INIT
- LOAD DATABASE
- LOAD TRANSACTION
- DUMP TRANSACTION
- BACKUP LOG
- RECONFIGURE
- RESTORE LOG
- UPDATE STATISTICS

Các ví dụ về transaction

Declare @tranname varchar(20)

Select @tranname ='MyTran'

Begin tran @tranname

Use Northwind

**Delete from [Order Details] where
OrderID=10248**

Commit tran @tranname

Select * from [Order Details] where OrderID=10248

Go

Quay lui trước những thay đổi

**ROLLBACK [TRAN[SACTION]
[transaction_name |savepoint_name]**

Dùng để quay ngược một transaction tởờng minh hay ngầm định về lại điểm bắt đầu, hay về điểm dừng (save-point) bên trong 1 transaction

Các ví dụ về transaction

Begin Tran

Use Northwind

Update Products

set UnitPrice =UnitPrice +10

where ProductName like 'A%'

If (Select MAX(unitprice) from Products

where ProductName like 'A%')>100

Begin

RollBack tran

Print 'Transaction rolled back'

End

Else

Begin

Commit Tran

Print 'Transaction committed'

End

Tạo điểm dừng cho một Transaction

- Lệnh SAVE TRANSACTION dùng để đặt 1 điểm dừng (save point) bên trong 1 transaction. Điểm dừng chia transaction thành 1 các phần khác nhau sao cho transaction có thể quay về lại điểm dừng này nếu 1 phần của transaction bị loại bỏ có điều kiện.
- Cú pháp
***SAVE TRAN[SACTION]
{savepoint_name }***

Các ví dụ về transaction

Thực thi 1 transaction với điểm dừng:

```
Select * from [Order Details] where ProductID in(3,7)
```

Begin Tran

Use Northwind

Update Products

set UnitsInStock =UnitsInStock+20

where ProductName like 'A%'

Update [Order Details]

set Discount =Discount+0.25

where ProductID in (3,7)

Save Tran tran1

Các ví dụ về transaction

Update [Order Details]

set UnitPrice =UnitPrice +10 where ProductID in (3,7)

Update [Order Details]

set Discount =Discount+0.5 where ProductID in (3,7)

if (Select discount from [Order Details] where ProductID In(3,7))<1

Begin

print 'Transaction 1 has been committed but transaction 2
has not been committed'

RollBack tran tran1

End

Else

Begin

print 'Both the transactions have been committed'

Commit Tran

End

select * from [Order Details] where ProductID in(3,7)

Các ví dụ về transaction

Dùng Transaction và cơ chế quản lý lỗi:

Begin Try

Begin tran

Update products

set UnitsInstock =100 where ProductID in (3,7)

Update [Order details]

set quantity =Quantity +100

where ProductID in(3,7)

Commit tran

End Try

Các ví dụ về transaction

Begin catch

Rollback tran

Raiserror ('Transaction Error',16,1)

Return

End catch

Hàm XACT_STATE

- Chỉ ra yêu cầu đang chạy hiện thời có transaction người dùng nào đang hoạt động không và transaction đó có thể được commit hay không?

1	Yêu cầu hiện thời có 1 transaction người dùng đang hoạt động có thể commit được
0	Không có transaction nào đang hoạt động
-1	Yêu cầu hiện thời có 1 transaction người dùng đang hoạt động nhưng có lỗi nên transaction được xem là uncommittable, không thể commit hay rollback về điểm dừng. Yêu cầu không thể “write” được cho đến khi transaction được rollback hoàn toàn.

Hàm XACT_STATE

Begin Try

Begin Tran

Delete From Products Where Productid =100

Commit Tran

End Try

Begin Catch

If (Xact_state())=-1

Begin

Print 'The Transaction Is In An Uncommittable
State' +'Rolling Back Transaction'

Rollback Tran

End

Hàm XACT_STATE

If (Xact_state())=1

Begin

Print 'The Transation Is Committable'
+'Committing Transaction'

Commit Tran

End

If (Xact_state())=0

Begin

Print 'No The Transation Is Committable'
+'Committing Transaction'

Rollback Tran

End

End Catch

Go

Điều khiển đồng thời

- ❑ Khái niệm: là sự kết hợp xử lý đồng thời những transaction trong 1 hệ CSDL đa người dùng
- ❑ Mục tiêu: đảm bảo sự tuần tự của các transaction để không gây nên các vấn đề về nhất quán và toàn vẹn dữ liệu sau đây:
 - Lost Updates
 - Uncommitted data
 - Inconsistent retrievals

Điều khiển đồng thời

Lost Updates (mất cập nhật):

- Các cập nhật sẽ bị mất khi hai hay nhiều giao tác chọn cùng 1 hàng và cùng cập nhật hàng đó.
- Các giao tác không biết về nhau. Cập nhật cuối cùng sẽ viết chồng lên các cập nhật được các giao tác khác thực hiện.

Điều khiển đồng thời

- **Lost Updates**
(Lịch tuần tự)

TRANSACTION

T1: cộng 0.5 điểm
T2: trừ 3 điểm

COMPUTATION

mark = mark + 0.5
mark = mark - 3

Time	Transaction	Step	Stored valued
1	T1	Đọc mark (read(mark))	6
2	T1	Mark = mark +0.5	
3	T1	Ghi mark (write(mark))	6.5
4	T2	Đọc mark (read(mark))	6.5
5	T2	Mark = mark - 3	
6	T2	Ghi mark (write(mark))	3.5

Điều khiển đồng thời

- **Lost Updates**

(Lịch đồng thời)

TRANSACTION

COMPUTATION

T1: cộng 0.5 điểm

mark = mark + 0.5

T2: trừ 3 điểm

mark = mark - 3

Time	Transaction	Step	Stored valued
1	T1	Đọc mark (read(mark))	6
2	T2	Đọc mark (read(mark))	6
3	T1	Mark = mark + 0.5	
4	T2	Mark = mark – 3	
5	T1	Ghi mark (write(mark))	6.5
6	T2	Ghi mark (write(mark))	3.0

Điều khiển đồng thời

- **Uncommitted data:** dữ liệu chưa được chuyển giao:

Xảy ra khi giao tác thứ 2 chọn 1 hàng đang được cập nhật bởi 1 giao tác khác. Giao tác thứ 2 đọc dữ liệu lúc chưa được công nhận (commit) và có thể bị thay đổi bởi giao tác đang thực hiện việc cập nhật.

Điều khiển đồng thời

- Uncommitted data

TRANSACTION

T1: cộng 0.5 điểm

T2: trừ 3 điểm

COMPUTATION

mark = mark + 0.5

mark = mark - 3

Time	Transaction	Step	Stored valued
1	T1	Đọc mark (read(mark))	6
2	T1	Mark = mark + 0.5	
3	T1	Ghi mark (write(mark))	6.5
4	T1	Rollback	
5	T2	Đọc mark (read(mark))	6.0
6	T2	Mark = mark - 3	
7	T2	Ghi mark (write(mark))	3.0

Điều khiển đồng thời

- Uncommitted data

Time	Transaction	Step	Stored valued
1	T1	Đọc mark (read(mark))	6
2	T1	Mark = mark + 0.5	
3	T1	Ghi mark (write(mark))	6.5
4	T2	Đọc mark (read(mark))	6.5
5	T2	Mark = mark – 3	3.5
6	T1	Rollback	
7	T2	Ghi mark (write(mark))	3.5

Điều khiển đồng thời

Inconsistent retrievals: truy xuất không nhất quán

- ❑ Xảy ra khi giao tác thứ 2 truy xuất cùng 1 hàng nhiều lần và dữ liệu mỗi lần đọc mỗi khác. Truy xuất không nhất quán tương tự như mỗi quan hệ chưa được chuyển giao, một giao tác khác đang thay đổi dữ liệu trong khi giao tác thứ hai đọc dữ liệu.

Điều khiển đồng thời

- Inconsistent retrievals**

TABLE 9.9 READ/WRITE CONFLICT SCENARIOS: CONFLICTING DATABASE OPERATIONS MATRIX

Operations	TRANSACTIONS		RESULT
	T1	T2	
	Read	Read	No conflict
	Read	Write	Conflict
	Write	Read	Conflict
	Write	Write	Conflict

Điều khiển đồng thời

- **Inconsistent retrievals**

T1	T2
SELECT sum(mark) From enroll WHERE SID = '142'	UPDATE enroll SET mark = mark +3 WHERE SID= '142' AND CID = 'C01'
	UPDATE enroll SET mark = mark - 3 WHERE SID= '142' AND CID = 'C02'

Tính cô lập và bài toán đồng thời (Isolation and Concurrency Problem)

- ❑ Khi 1 số user chỉnh sửa dữ liệu có thể làm ảnh hưởng đến các user khác đang xem hay chỉnh sửa cùng dữ liệu. Ta gọi các user này đang truy xuất dữ liệu đồng thời (accessing the data concurrently)
- ❑ Nếu DBMS không kiểm soát tính đồng thời (concurrency control) thì sẽ xảy ra các hiệu ứng sau:
 - Mất cập nhật (Lost updates).
 - Đọc dữ liệu chưa được chuyển giao (Uncommitted read).
 - Đọc dữ liệu bị sai lệch (Non-repeatable read).
 - Đọc dữ liệu ảo (Phantom reads)

Lost update

reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A



App B



Lost update

reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

**update reservations
set name = 'John'
where seat = '7C'**

App B

The diagram illustrates a lost update scenario. At the top, a table titled 'reservations' has three columns: 'seat', 'name', and '...'. The first row has '7C' in the 'seat' column and an empty 'name' column. The second row has '7B' in the 'seat' column and an empty 'name' column. The third row has '...' in the 'seat' column and an empty 'name' column. Below the table, two arrows point upwards towards the 'name' column. The left arrow originates from a box labeled 'App A' which contains the SQL statement: 'update reservations set name = 'John' where seat = '7C''. The right arrow originates from a box labeled 'App B' which is currently empty.

Lost update

reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

**update reservations
set name = 'John'
where seat = '7C'**

App B



Lost update

reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

update reservations
set name = 'John'
where seat = '7C'

App B

update reservations
set name = 'Mary'
where seat = '7C'



Lost update

reservations

seat	name	...
7C	Mary	
7B	_____	
...		

App A

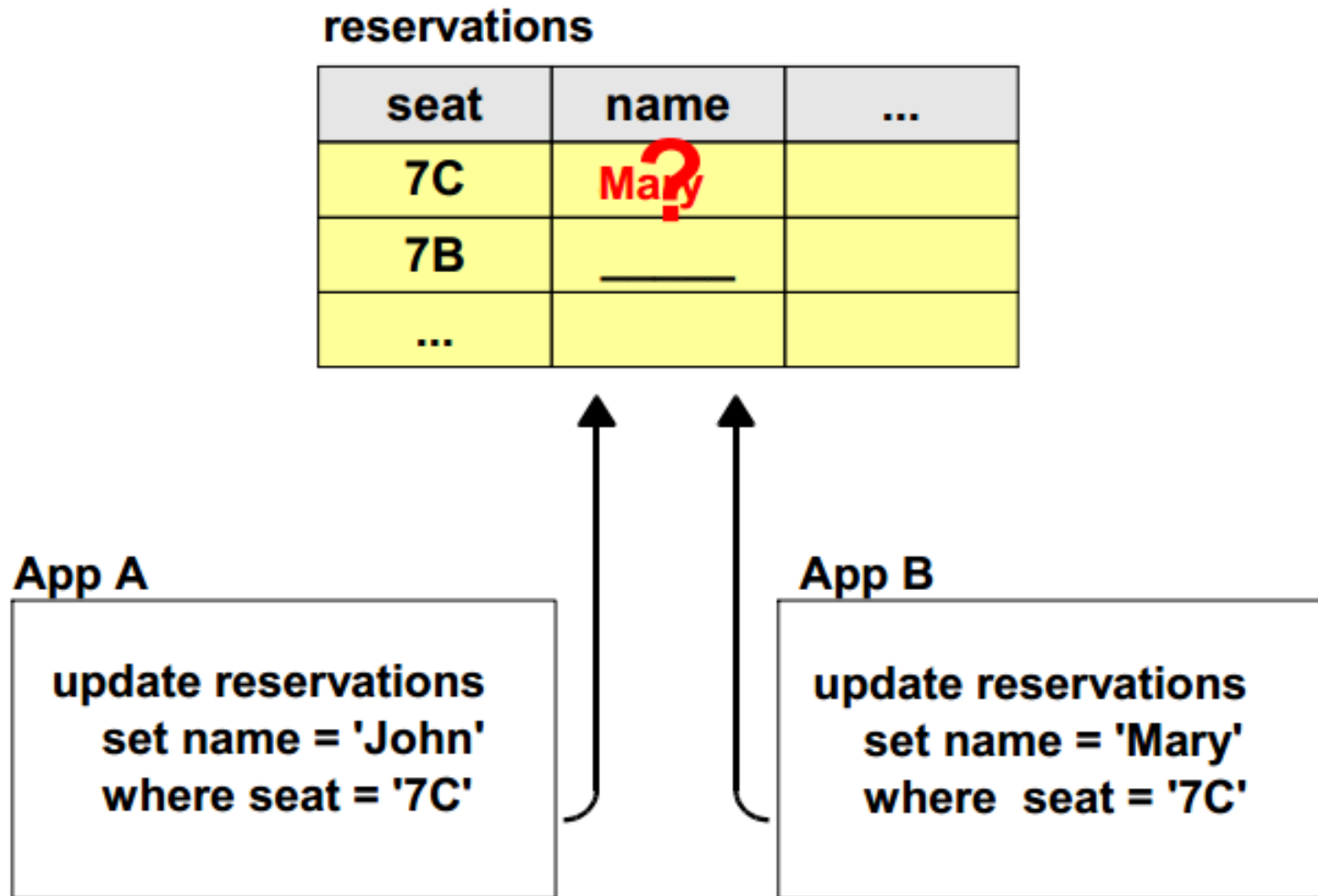
update reservations
set name = 'John'
where seat = '7C'

App B

update reservations
set name = 'Mary'
where seat = '7C'



Lost update



The first update from application A was a “lost update”

Uncommitted read (also known as “dirty read”)

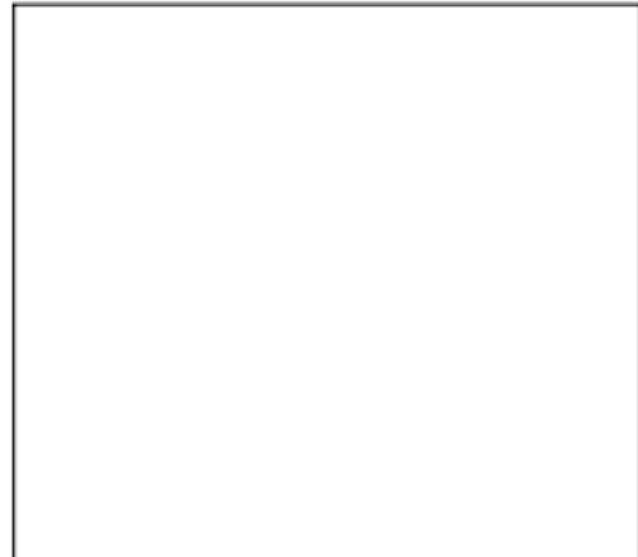
reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

A large, empty rectangular box with a black border, representing the workspace for App A.

App B

A large, empty rectangular box with a black border, representing the workspace for App B.

Uncommitted read (also known as “dirty read”)

reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

**update reservations
set name = 'John'
where seat = '7C'**

App B



Uncommitted read (also known as “dirty read”)

reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

**update reservations
set name = 'John'
where seat = '7C'**

App B



Uncommitted read (also known as “dirty read”)

reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

**update reservations
set name = 'John'
where seat = '7C'**

App B

**Select name
from reservations
where seat is '7C'**

Uncommitted read (also known as “dirty read”)

reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

update reservations
set name = 'John'
where seat = '7C'

App B

Select name
from reservations
where seat is '7C'

John

Uncommitted read (also known as “dirty read”)

reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

update reservations
set name = 'John'
where seat = '7C'

Roll back

App B

Select name
from reservations
where seat is '7C'

John

Uncommitted read (also known as “dirty read”)

reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

**update reservations
set name = 'John'
where seat = '7C'**

Roll back

App B

**Select name
from reservations
where seat is '7C'**

John

Uncommitted read (also known as “dirty read”)

reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

**update reservations
set name = 'John'
where seat = '7C'**

Roll back

App B

**Select name
from reservations
where seat is '7C'**

John

**Further processing in App
B uses incorrect /
uncommitted value of
“John”**

Non-repeatable read

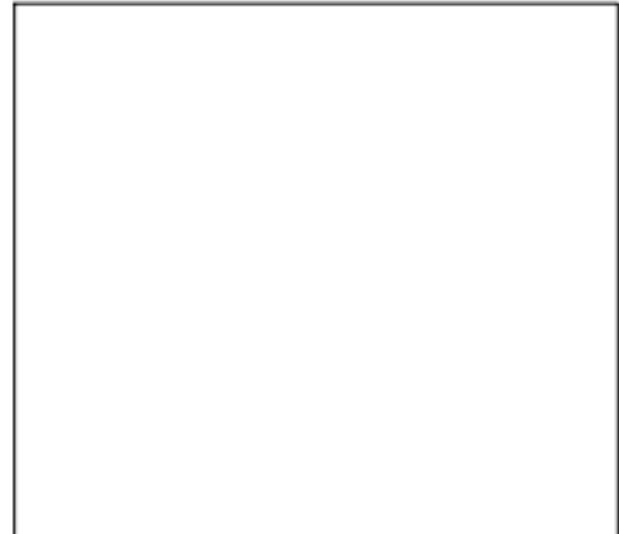
reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

A large, empty rectangular box with a black border, intended for App A to store or process data.

App B

A large, empty rectangular box with a black border, intended for App B to store or process data.

Non-repeatable read

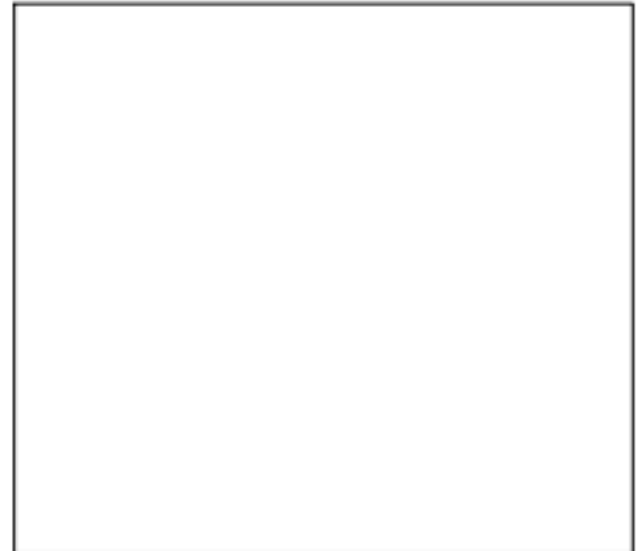
reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

```
select seat
from reservations
where name is NULL
```

App B



Non-repeatable read

reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

**select seat
from reservations
where name is NULL**

App B

7C
7B

Non-repeatable read

reservations

seat	name	...
7C	_____	
7B	_____	
...		

7C
7B

App A

```
select seat
from reservations
where name is NULL
```

App B

```
update reservations
set name = 'John'
where seat = '7C'
```

Non-repeatable read

reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

**select seat
from reservations
where name is NULL**

App B

**update reservations
set name = 'John'
where seat = '7C'**

7C

7B

Non-repeatable read

reservations

seat	name	...
7C	John	
7B	_____	
...		

7C
7B

App A

```
select seat
from reservations
where name is NULL

...
select seat
from reservations
where name is NULL
```

App B

```
update reservations
set name = 'John'
where seat = '7C'
```

Non-repeatable read

reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

**select seat
from reservations
where name is NULL**

...
**select seat
from reservations
where name is NULL**

App B

**update reservations
set name = 'John'
where seat = '7C'**

7C

7B

7B

Non-repeatable read

reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

select seat
from reservations
where name is NULL

...
select seat
from reservations
where name is NULL

App B

update reservations
set name = 'John'
where seat = '7C'

The same SELECT (read) returns a different result: Less rows (in this case '7C' doesn't show anymore). This is a non-repeatable read

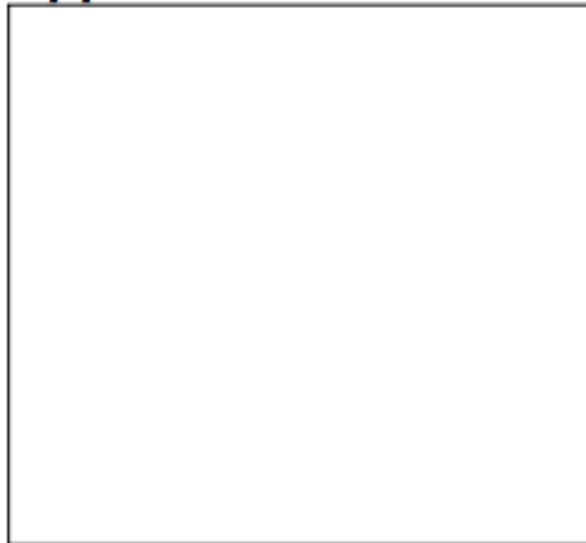
So it's not a repeatable read: Issuing the exact same SELECT statement within a transaction leads to different results

Phantom read

reservations

seat	name	...
7C	Susan	
7B	_____	
...		

App A

A large, empty rectangular box with a black border, intended for App A.

App B

A large, empty rectangular box with a black border, intended for App B.

Phantom read

reservations

seat	name	...
7C	Susan	
7B	_____	
...		

App A

```
select seat  
from reservations  
where name is NULL
```

App B



Phantom read

reservations

seat	name	...
7C	Susan	
7B	_____	
...		

App A

select seat
from reservations
where name is NULL

7B

App B



Phantom read

reservations

seat	name	...
7C	Susan	
7B	_____	
...		

App A

**select seat
from reservations
where name is NULL**

7B

App B

**update reservations
set name = NULL
where seat = '7C'**

Phantom read

reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

**select seat
from reservations
where name is NULL**

App B

**update reservations
set name = NULL
where seat = '7C'**

7B

Phantom read

reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

```
select seat
from reservations
where name is NULL
```

```
...
select seat
from reservations
where name is NULL
```

App B

```
update reservations
set name = NULL
where seat = '7C'
```

7B

Phantom read

reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

**select seat
from reservations
where name is NULL**

...

**select seat
from reservations
where name is NULL**

App B

**update reservations
set name = NULL
where seat = '7C'**

7B

7C

7B

Phantom read

reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

select seat
from reservations
where name is NULL

...

select seat
from reservations
where name is NULL

App B

update reservations
set name = NULL
where seat = '7C'

The same SELECT (read) returns a different result: More rows (phantom rows, in this case '7C', is shown)
This is a phantom read

7B

7C

7B

Các vấn đề

- **Lost Update (cập nhập mất dữ liệu):** xảy ra khi 2 hay nhiều transaction chọn cùng một dữ liệu và sau đó cập nhập dòng dựa trên giá trị cũ.
- **Uncommitted Dependency (Dirty Read: đọc dữ liệu sai) :** xảy ra khi Transaction thứ hai chọn một dòng đang sẵn sàng cập nhật bởi một transaction khác.
- **Inconsistent Analysis (Nonrepeatable Read: đọc dữ liệu hai lần) :** xảy ra khi transaction thứ 2 truy xuất cùng một dữ liệu với vài lần đọc, với những dữ liệu khác nhau ở mỗi lần đọc.
- **Phantom Reads(đọc các mẫu tin ma):** Xảy ra khi hành động Insert hay delete được thi hành trên một dòng dữ liệu mà nó thuộc vùng dữ liệu đọc của một transaction khác.

Khóa (Locks)

- Là cơ chế ngăn chặn các mâu thuẫn do các user không thể đọc hay hiệu chỉnh các dữ liệu mà các dữ liệu này hiện đang mở một tiến trình xử lý khác.
- Tuy nhiên, bạn vẫn có thể thao tác trên những đối tượng còn phụ thuộc vào chuyển tác user khác đang thực hiện. Khi đó hệ thống sẽ kiểm soát xem tiến trình của bạn có tương thích với quá trình trước đó hay không.

Transaction và cơ chế khóa

- ❑ Trước khi transaction đọc hay hiệu chỉnh dữ liệu, nó cần được bảo vệ tránh ảnh hưởng các transaction khác đang chỉnh sửa cùng dữ liệu.
- ❑ Transaction yêu cầu khóa trên dữ liệu đang dùng
- ❑ Có nhiều chế độ khóa khác nhau phụ thuộc vào mức độ, phụ thuộc dữ liệu của transaction.
- ❑ Sẽ không có transaction nào được cấp khóa nếu gây mâu thuẫn với chế độ khóa đã được cấp trên cùng dữ liệu cho một transaction khác trước đó

Transaction và cơ chế khóa

- ❑ Nếu transaction yêu cầu 1 chế độ khóa có tranh chấp, Database Engine sẽ bắt transaction này dừng (pause) cho đến khi khóa trước đó được giải phóng.
- ❑ Tất cả các khóa sẽ được giải phóng khi transaction hoàn thành (bằng commit hay roll back)

Transaction và cơ chế khóa

- ❑ Các ứng dụng không trực tiếp yêu cầu khóa. Các khóa được quản lý nội bộ bởi lock manager (bộ quản lý khóa- 1 thành phần của DB Engine)
- ❑ Khi Database Engine xử lý 1 lệnh Transact-SQL, query processor (bộ xử lý truy vấn) sẽ xác định tài nguyên nào được truy xuất, loại khóa nào cần dùng, thiết lập mức cô lập cho transaction. Kế đến query processor yêu cầu 1 khóa phù hợp từ lock manager. Lock manager cấp khóa nếu không có mâu thuẫn.

Các loại khóa

- ✓ **Shared locks** (khóa chia sẻ/ dùng chung /Khóa đọc): được dùng cho những thao tác không làm thay đổi hay cập nhật dữ liệu như lệnh Select.
- ✓ **Exclusive locks** (khóa độc quyền/ khóa ghi) : được dùng cho những thao tác hiệu chỉnh dữ liệu như Insert, Update, Delete.
- ✓ **Update locks** : dùng trên những tài nguyên có thể cập nhật.
- ✓ **Insert Locks** : Dùng để thiết lập một Lock kế thừa.
- ✓ **Scheme Locks** : được dùng khi thao tác (thuộc giản đồ của Table) đang thực thi.
- ✓ **Bulk Update locks** : Cho phép chia sẻ cho Bulk-copy thi hành.
- ✓ **Deadlock** (tắc nghẽn): xảy ra khi có sự phụ thuộc chu trình giữa hai hay nhiều luồng cho một tập hợp tài nguyên nào đó

Scheduler

- ❑ Khái niệm: là 1 chương trình DBMS thiết lập thứ tự các thao tác trong những transaction đồng thời
- ❑ Các phương pháp:
 - Locking (khóa)
 - Time Stamping (nhãn thời gian)
 - Optimistic

Phương pháp Locking

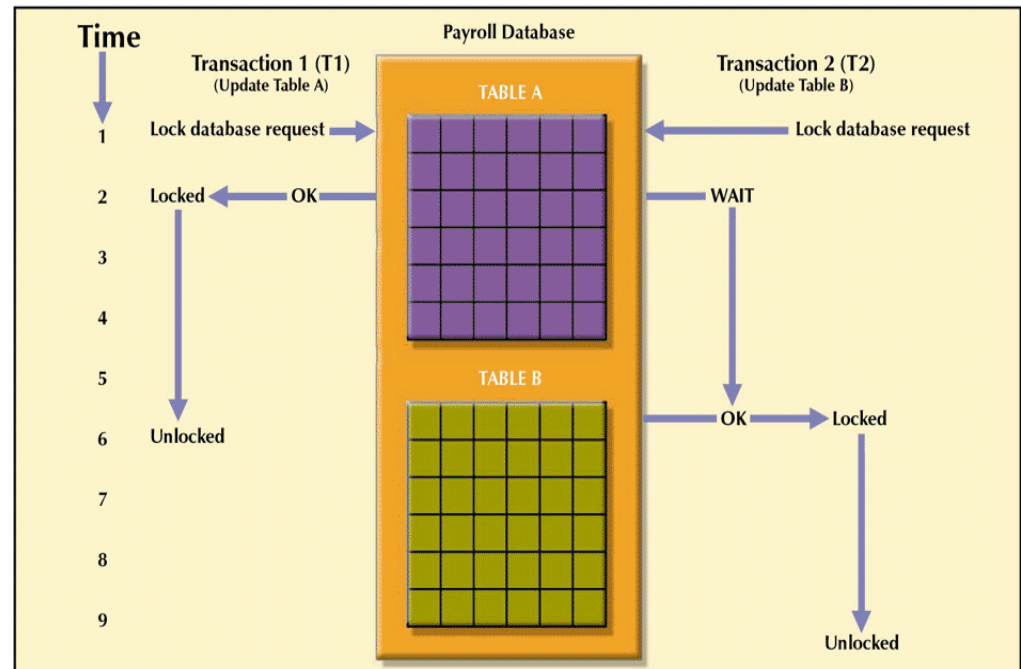
- ❑ Dùng để điều khiển đồng thời:
 - 1 khóa được cấp để sử dụng “độc quyền” 1 hạng mục dữ liệu trong transaction hiện hành.
 - 1 transaction được cấp khóa trước khi truy cập dữ liệu; sau khi transaction hoàn tất, khóa phải được giải phóng
 - Lock manager quản lý những thông tin về khóa.

Phương pháp Locking (tt)

❑ Các mức Locking

- Database level
- Table level
- Page level
- Row level
- Field (attribute) level

FIGURE 9.3 A DATABASE-LEVEL LOCKING SEQUENCE



Phương pháp Locking (tt)

❑ Các kiểu khóa

– Binary Locks

- Có 2 trạng thái: locked (1) hoặc unlocked (0).
- Nếu 1 object bị lock bởi 1 transaction, không transaction nào được sử dụng object đó
- Nếu 1 object là unlocked, bất kỳ transaction nào cũng có thể lock object đó để sử dụng
- 1 transaction phải “unlock” object sau khi hoàn tất.

Phương pháp Locking (tt)

❑ Các kiểu lock:

– Exclusive Locks

- Tồn tại khi transaction ghi dữ liệu
- Được sử dụng khi có khả năng độ dữ liệu.
- Một exclusive lock (khóa độc quyền) sẽ được gán khi transaction muốn ghi dữ liệu và dữ liệu đó chưa bị lock
- Được dùng cho thao tác sửa đổi dữ liệu như lệnh INSERT, UPDATE hay DELETE. Bảo đảm nhiều lệnh cập nhật không thực hiện trên cùng 1 tài nguyên cùng 1 lúc

Phương pháp Locking (tt)

❑ Các kiểu lock

– Exclusive Locks

- Ví dụ: nếu lệnh Update sửa đổi các hàng trong một bảng mà lệnh này có kết nối (join) với 1 bảng khác thì sẽ cần bao nhiêu khóa?
- Một khóa shared (dùng chung) cho các hàng đọc được trong bảng kết nối
- Một khóa exclusive (độc quyền) cho các hàng được cập nhật trong bảng update.

Phương pháp Locking (tt)

❑ Các kiểu lock

– Shared Locks

- Một shared lock tồn tại khi các transaction đồng thời đọc dữ liệu
- Một shared lock không làm đọng độ dữ liệu khi các transaction đồng thời chỉ đọc dữ liệu
- Một shared lock được gán khi transaction muốn đọc dữ liệu và dữ liệu đó không tồn tại exclusive lock.

Phương pháp Locking (tt)

□ Các kiểu lock

– Intent Locks

- DB Engine dùng khóa này để bảo vệ việc đặt khóa S (Shared) hay X (eXclusive) trên tài nguyên ở mức thấp hơn. Các khóa này luôn luôn được tạo trước khi khóa ở mức thấp hơn được đặt, nhằm báo hiệu có khóa mức thấp hơn.
- Các loại khóa intent là: Intent Shared (IS), Intent eXclusive (IX) và Shared with Intent eXclusive (SIX).

Phương pháp Locking (tt)

❑ Các kiểu lock

– Intent Locks

- Ví dụ: Khóa IS được yêu cầu ở mức bảng trước khi khóa S được yêu cầu ở 1 trang hay hàng bên trong bảng. Nhờ khóa IS ở mức bảng sẽ ngăn không cho các transaction khác đặt khóa X trên bảng này, cải thiện được việc thực thi vì khi đó DB engine chỉ cần khảo sát khóa intent ở mức bảng là có thể xác định 1 transaction khác có thể chiếm được 1 khóa trên bảng đó hay không, không cần phải tìm từng khóa trên mỗi trang hay mỗi hàng của bảng đó.

Phương pháp Locking (tt)

- ❑ Hai vấn đề với phương pháp locking
 - Schedule (lịch) của transaction không khả tuần tự
 - Có thể tạo ra deadlock
- ❑ Giải pháp:
 - Khả tuần tự: two phase locking (khóa 2 giai đoạn)
 - Deadlock: phát hiện và ngăn chặn

Phương pháp Locking (tt)

❑ Two-phase Locking (giao thức khóa 2 giai đoạn)

- Giao thức **two-phase locking** xác định cách transaction có được khóa và giải phóng khóa, đảm bảo được tính khả tuần tự (**serializability**) nhưng không tránh được deadlock
- Giai đoạn **growing (cấp khóa)**: transaction lấy được tất cả các khóa cần thiết nhưng không khóa dữ liệu. Tất cả các khóa được đặt vào **locked point**.
- Giai đoạn **shrinking (thu hồi khóa, trả khóa)**: transaction giải phóng tất cả các khóa và không lấy thêm khóa mới nào

Phương pháp Locking (tt)

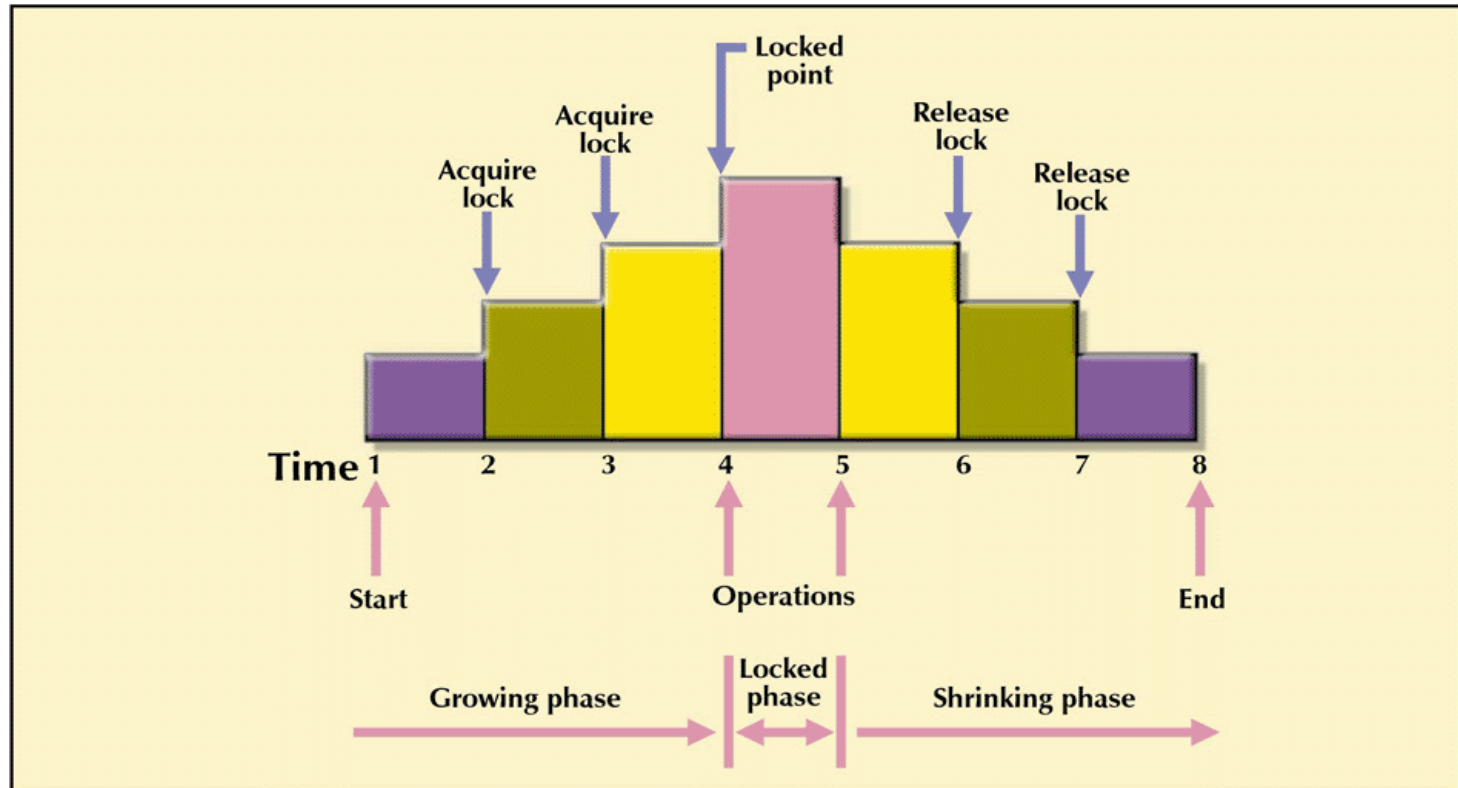
❑ Locking two-phase

- Quy tắc cho giao thức Two-Phase Locking
 - Không có 2 transaction nào có khóa đụng độ (mâu thuẫn)
 - Trong cùng 1 transaction không có thao tác không khóa nào đi trước thao tác có khóa.
 - Không có dữ liệu nào bị ảnh hưởng cho đến khi tất cả các khóa có được.

Phương pháp Locking (tt)

❑ Locking two-phase

FIGURE 9.7 TWO-PHASE LOCKING PROTOCOL



Phương pháp Locking (tt)

Nghẽn khóa-Deadlock:

- ❑ Là một hoàn cảnh trong đó 2 user (hay transaction) có các khóa trên các đối tượng khác nhau và mỗi user đang chờ khóa trên đối tượng của người dùng khác
- ❑ Deadlock còn được gọi là deadly embrace

Phương pháp Locking (tt)

Nghẽn khóa-Deadlock

- ❑ Khi bị nghẽn khóa, các chương trình ứng dụng không thể giải quyết bế tắc này. DBMS phải phát hiện thấy và phải giải quyết gỡ bỏ nghẽn khóa.
- ❑ Chỉ có 1 cách là hủy bỏ (quay lui) một hay nhiều giao tác để giải quyết bế tắc.
- ❑ Người dùng không nhận thấy được sự xuất hiện của tình trạng nghẽn khóa, DBMS phải tự động khởi động hay hủy bỏ một hay một số thao tác

Phương pháp Locking (tt)

❑ Deadlock

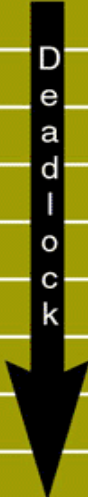
- 2 transactions cùng đợi để unlock dữ liệu
- Deadlocks tồn tại khi transactions T1 và T2 :
 - T1 = Truy cập đề mục dữ liệu X và Y
 - T2 = Truy cập đề mục dữ liệu Y và X
- Nếu T1 không unlock dữ liệu Y, T2 không thể bắt đầu; và nếu T2 không unlock dữ liệu X, T1 không thể tiếp tục.

Phương pháp Locking (tt)

❑ Deadlock

TABLE 9.11 HOW A DEADLOCK CONDITION IS CREATED

TIME	TRANSACTION	REPLY	LOCK STATUS	
			Data X	Data Y
0				
1	T1:LOCK(X)	OK	Unlocked	Unlocked
2	T2: LOCK(Y)	OK	Locked	Unlocked
3	T1:LOCK(Y)	WAIT	Locked	Locked
4	T2:LOCK(X)	WAIT	Locked	Locked
5	T1:LOCK(Y)	WAIT	Locked	Locked
6	T2:LOCK(X)	WAIT	Locked	Locked
7	T1:LOCK(Y)	WAIT	Locked	Locked
8	T2:LOCK(X)	WAIT	Locked	Locked
9	T1:LOCK(Y)	WAIT	Locked	Locked
...
...
...
...



Phương pháp Locking (tt)

❑ Deadlock: 3 kỹ thuật để giải quyết deadlocks:

- **Ngăn chặn Deadlock:** Một transaction sẽ bị từ chối/ hủy nếu yêu cầu lock mới và lock mới này có khả năng gây nên dealock. Sau đó transaction sẽ được khởi động lại
- **Phát hiện Deadlock:** DBMS định kỳ kiểm tra deadlocks. Nếu có deadlock, một trong những transaction phải bị hủy để transaction kia tiếp tục
- **Tránh Deadlock:** Transaction phải lấy được tất cả các khóa nó cần trước khi thực thi

Cơ chế khóa nhiều cấp (Lock granularity)

- ❑ Cho phép transaction khóa các loại tài nguyên khác nhau.
- ❑ Để giảm chi phí khóa, Db engine khóa tự động tài nguyên tùy theo cấp độ của nhiệm vụ.
- ❑ Khóa ở mức nhỏ hơn như hàng, làm tăng khả năng đồng thời nhưng lại làm tăng chi phí vì cần nhiều khóa hơn khi có nhiều hàng cần khóa.
- ❑ Khóa ở mức lớn hơn, như mức bảng, thực thi đồng thời sẽ khó khăn hơn vì khi cả bảng được khóa sẽ hạn chế việc truy xuất đến các phần của bảng của các transaction khác. Nhưng chi phí sẽ thấp vì cần dùng ít khóa hơn.

Lock granularity

- ❑ DB Engine thường thực hiện nhiều mức khóa khác nhau để bảo vệ đầy đủ tài nguyên.
- ❑ Khóa ở nhiều mức khác nhau được gọi là lock hierarchy.
- ❑ Ví dụ: để bảo vệ đầy đủ việc đọc 1 index, DB Engine có thể phải chiếm các khóa shared trên các hàng, và khóa intent shared trên các trang và bảng

Tài nguyên có thể bị khóa

Resource	Description
RID	A row identifier used to lock a single row within a heap.
KEY	A row lock within an index used to protect key ranges in serializable transactions.
PAGE	An 8-kilobyte (KB) page in a database, such as data or index pages.
EXTENT	A contiguous group of eight pages, such as data or index pages.
HoBT	A heap or B-tree. A lock protecting a B-tree (index) or the heap data pages in a table that does not have a clustered index.
TABLE	The entire table, including all data and indexes.
FILE	A database file.
APPLICATION	An application-specified resource.
METADATA	Metadata locks.
ALLOCATION_UNIT	An allocation unit.
DATABASE	The entire database.

Sự tương thích khóa (Lock compatibility)

- ❑ Tương thích khóa dùng để kiểm soát nhiều transaction có được chiếm các khóa trên cùng tài nguyên cùng lúc hay không?
- ❑ Nếu tài nguyên đã bị khóa bởi 1 transaction, một yêu cầu khóa mới có thể được cấp chỉ khi chế độ của khóa được yêu cầu tương thích với chế độ của khóa hiện có.
- ❑ Nếu không tương thích với khóa hiện có, transaction yêu cầu khóa mới sẽ đợi cho đến khi khóa hiện tại được giải phóng hay hết thời gian đợi (time-out).

Sự tương thích khóa

Ví dụ:

- ❑ Không có chế độ khóa nào tương thích với khóa exclusive. Trong khi đang có khóa exclusive(X) thì không transaction nào có thể chiếm được bất kỳ loại khóa nào (shared, update hay exclusive) trên tài nguyên đó cho đến khi khóa exclusive được giải phóng.
- ❑ Nếu khóa shared(S) đang được dùng trên tài nguyên, các transaction khác có thể chiếm các khóa shared hay khóa update(U) ngay trên tài nguyên đó ngay cả khi transaction đầu chưa hoàn tất. Tuy nhiên các transaction không thể có được khóa exclusive cho đến khi khóa shared được giải phóng.

Sự tương thích khóa

Ví dụ: Sự tương thích khóa (Y:chấp nhận, N:không)

	Existing granted mode					
Requested mode	IS	S	U	IX	SIX	X
Intent shared (IS)	Yes	Yes	Yes	Yes	Yes	No
Shared (S)	Yes	Yes	Yes	No	No	No
Update (U)	Yes	Yes	No	No	No	No
Intent exclusive (IX)	Yes	No	No	Yes	No	No
Shared with intent exclusive (SIX)	Yes	No	No	No	No	No
Exclusive (X)	No	No	No	No	No	No

Cách sử dụng khóa

- ❑ Mặc định các transaction isolation là read committed, nghĩa là SQL Server bảo đảm chỉ có dữ liệu nào đã commit thì mới được đọc.
- ❑ Trong khi 1 hàng đang được cập nhật, dữ liệu chưa được commit, SQL Server sẽ buộc các transaction muốn đọc dữ liệu phải đợi cho đến khi dữ liệu được commit.

Cách sử dụng khóa

Ví dụ về cách sử dụng khóa:

- User1 đang thực hiện các lệnh sau để cập nhật điểm và ngày thi cho ứng viên có mã là '000002' trong bảng ExternalCandidate.

```
BEGIN TRANSACTION
```

```
UPDATE ExternalCandidate
```

```
SET siTestScore = 90
```

```
WHERE cCandidateCode='000002'
```

```
UPDATE ExternalCandidate
```

```
SET dTestDate = getdate()
```

```
WHERE cCandidateCode = '000002'
```

Cách sử dụng khóa

Ví dụ về cách sử dụng khóa:

- Trong khi transaction trên đang thực hiện, User2 muốn lập lịch phỏng vấn cho các ứng viên, nhưng không thể xem chi tiết của các ứng viên có điểm thi trên 80. User2 đang sử dụng các lệnh sau :

```
BEGIN TRANSACTION
```

```
SELECT * from ExternalCandidate
```

```
WHERE siTestScore > 80
```

```
UPDATE ExternalCandidate
```

```
SET dInterviewDate = getdate()+ 2
```

```
WHERE siTestScore > 80
```

Hãy xác định tại sao user2 không thể thực thi transaction

Cách sử dụng khóa

Ví dụ về cách sử dụng khóa:

- Các bảng sẽ bị khoá khi transaction trên máy 1 đang thực hiện.
- Khi transaction trên máy 1 kết thúc bằng cách dùng lệnh sau:

COMMIT TRANSACTION

Thì transaction trên máy 2 mới được thực hiện.

Các mức cô lập transaction

- ❑ Chọn mức cô lập cho transaction sẽ không làm ảnh hưởng đến các khóa đang có để tránh dữ liệu bị sửa đổi.
- ❑ Một transaction luôn nhận khóa độc quyền trên bất kỳ dữ liệu nào nó sửa đổi và sẽ giữ khóa cho đến khi transaction hoàn tất.
- ❑ Mức cô lập càng thấp thì người dùng càng có nhiều khả năng truy xuất dữ liệu đồng thời, nhưng cũng gây ra nhiều ảnh hưởng tương tranh và ngược lại

Các mức cô lập transaction

- ❑ Để chọn mức cô lập thích hợp thì phải cân đối giữa yêu cầu bảo toàn dữ liệu của ứng dụng với chi phí của mỗi mức cô lập.
- ❑ Mức cô lập cao nhất là tuần tự hóa (serializable)
- ❑ Mức cô lập thấp nhất là cho phép đọc dữ liệu chưa được commit (read uncommitted)

Bốn mức cô lập theo chuẩn ISO

1. **Read uncommitted:** mức thấp nhất, transaction bị cô lập chỉ đủ để đảm bảo các dữ liệu bị lỗi vật lý không được đọc thôi
2. **Read committed:** mức mặc định của DB Engine
3. **Repeated read**
4. **Serializable (khả tuần tự):** mức cao nhất, các transaction hoàn toàn bị cô lập khỏi các transaction khác

Các mức cô lập của SQL Server DB Engine

1. **Read uncommitted:** các lệnh có thể đọc các hàng bị ảnh hưởng bởi các transaction khác dù chưa được commit
2. **Read committed:** các lệnh không thể đọc dữ liệu đã bị sửa đổi nhưng chưa commit bởi các transaction khác
3. **Repeated read:** các lệnh không thể đọc dữ liệu đã bị sửa đổi bởi các transaction khác và không có transaction nào có thể sửa đổi dữ liệu đã được đọc bởi transaction hiện hành cho đến khi transaction hiện hành hoàn tất.

Các mức cô lập của SQL Server DB Engine

4. **Snapshot:** dữ liệu được đọc bởi bất kỳ lệnh nào trong 1 transaction thì sẽ được giữ giống như lúc bắt đầu transaction.
5. **Serializable**

Lệnh DBCC USEROPTIONS

- ❑ Để xác định mức cô lập hiện hành, dùng lệnh DBCC USEROPTIONS:

USE QLBH

GO

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

GO

DBCC USEROPTIONS

GO

Phạm vi của mức cô lập

- ❑ Khi mức cô lập được xác định, khóa dùng tất cả lệnh DML trong phiên làm việc đó sẽ theo mức cô lập này.
- ❑ Mức cô lập này duy trì cho đến khi phiên làm việc kết thúc hay mức cô lập được cài đặt mức mới.

Lệnh thay đổi mức cô lập

- ❑ SET TRANSACTION ISOLATION LEVEL
 {READ UNCOMMITTED
 | READ COMMITTED
 | REPEATABLE READ
 | SNAPSHOT
 | SERIALIZABLE }
 [:]

Lệnh thay đổi mức cô lập

❑ Ví dụ:

```
SET TRANSACTION ISOLATION LEVEL  
    REPEATABLE READ  
GO  
BEGIN TRANSACTION  
    SELECT * FROM ORDERS  
    SELECT * FROM CUSTOMERS  
COMMIT TRANSACTION
```