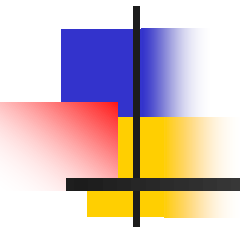
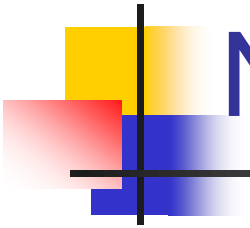


Managing Transactions, Locks



Nội dung

- Định nghĩa Transactions
- Thuộc tính của Transactions
- Các loại Transaction
- Công dụng Transaction log
- Định nghĩa Lock
- Các vấn đề đồng thời (Concurrency problem).
- Các kiểu Lock

Transaction

- Giao dịch là một đơn vị xử lý nguyên tố bao gồm một hoặc nhiều lệnh thực hiện đồng thời các lệnh tương tác đến CSDL
- Các lệnh trong một giao dịch thực hiện theo nguyên tắc hoặc tất cả đều thành công hoặc một lệnh thất bại thì toàn bộ giao dịch thất bại.
- Nếu việc thực thi một lệnh nào đó bị thất bại thì dữ liệu phải rollback trạng thái ban đầu

Ví dụ : Ngân hàng thực hiện chuyển tiền từ tài khoản A sang tài khoản B, cần thực hiện hai công việc :

- trừ tiền của tài khoản A,
- tăng tiền của tài khoản B

Transaction

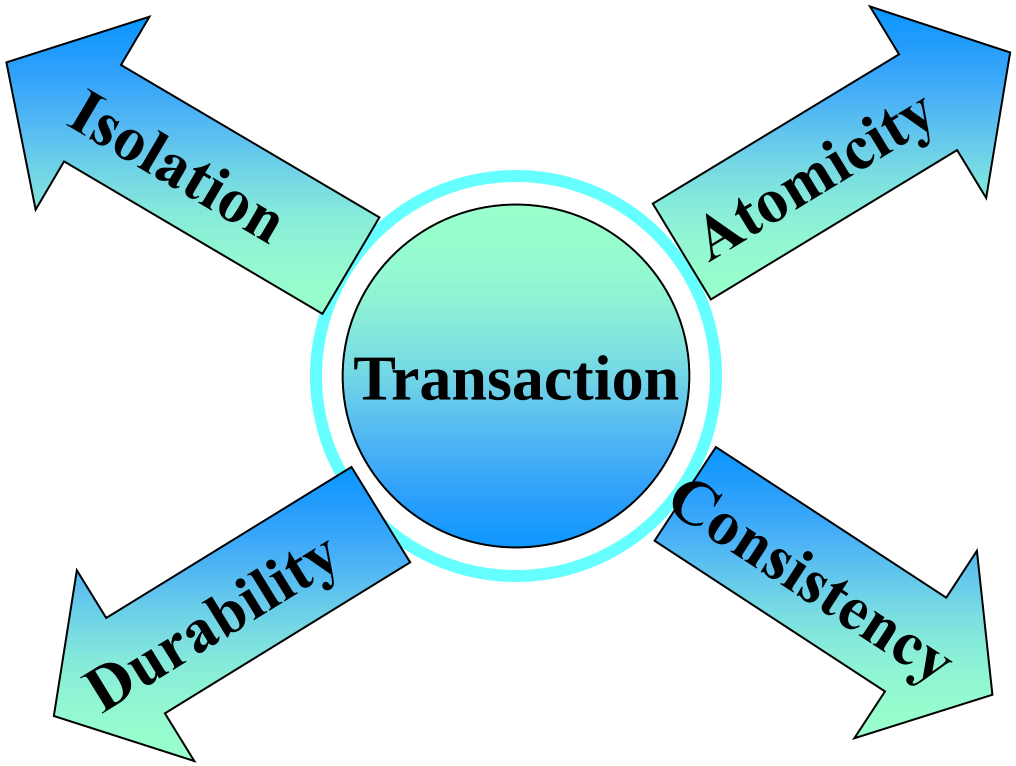
VD: giao dịch chuyển khoản 50$ từ tài khoản A sang tài khoản B

read(*A*)
A := *A* – 50
write(*A*)
read(*B*)
B := *B* + 50
write(*B*)

INSERT
UPDATE
DELETE
UPDATE
INSERT

Transaction

Transaction Properties



Thuộc tính của Transaction

- **Tính nguyên tử (Atomicity):** nếu một lệnh nào đó trong transaction bị fail thì dữ liệu trở lại trạng thái ban đầu.
- **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 giao dịch thành công kết quả của giao dịch được duy trì trong CSDL.

Các thuộc tính của Transaction - ACID

- Để đảm bảo tính Atomic của giao dịch, người dùng cần điều khiển tường minh việc Rollback của giao dịch.
- Kiểm tra lỗi khi thực hiện mỗi lệnh trong giao dịch để xử lý rollback.
- **Ví dụ**: viết thủ tục chuyển tiền từ tài khoản này đến tài khoản khác

  taikhoan (MaTK, TenTK, SoduTK)

Các thuộc tính của Transaction - ACID

Ví dụ:

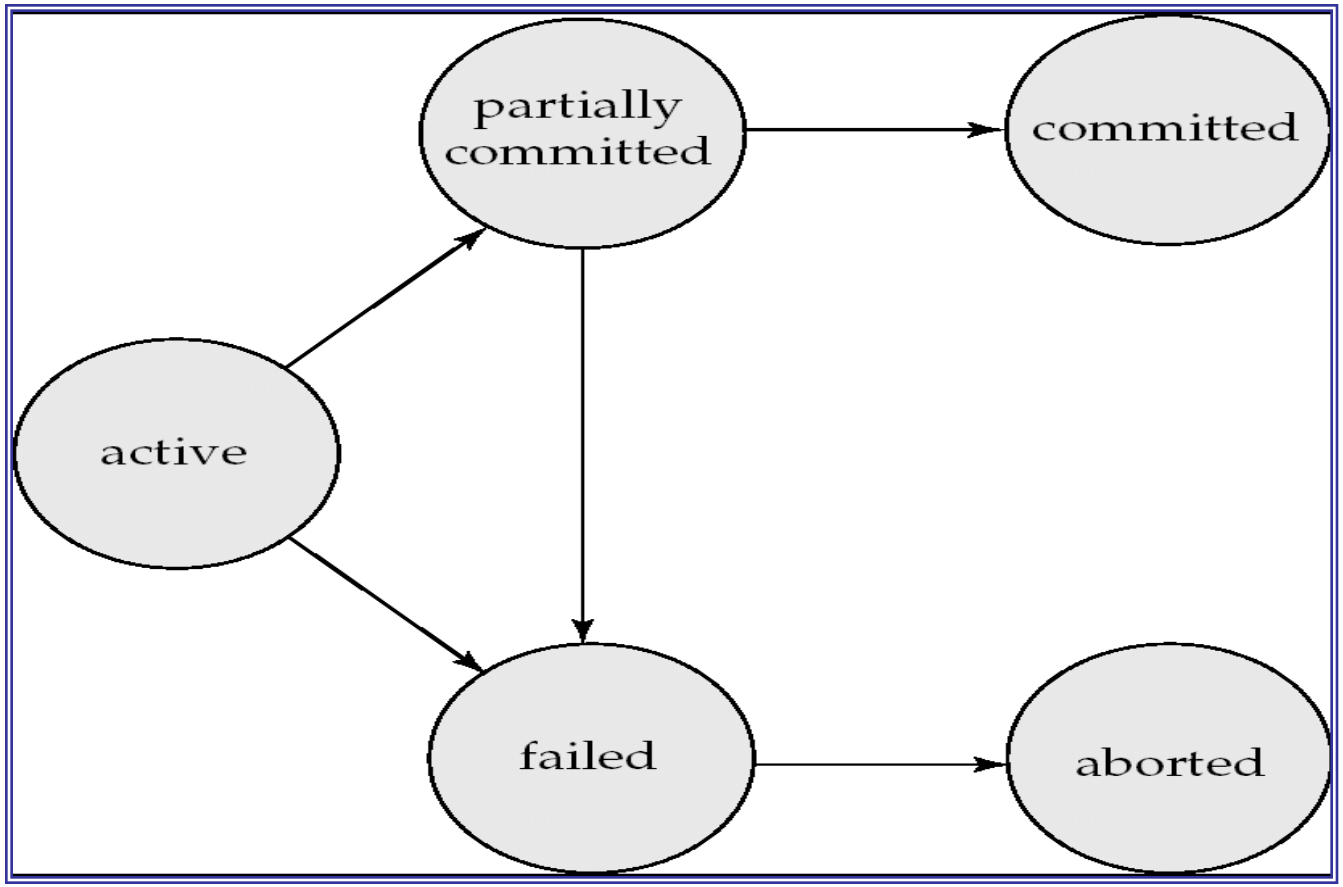
```
create proc usp_ChuyenKhoan
  @tkdi char(10), @tkden char(10), @sotien int
as
begin
    begin try
        BEGIN TRAN
            SET XACT_ABORT ON
            update TaiKhoan
            set SoDuTK=SoDuTK-@sotien
            where MaTK= @tkdi

            update TaiKhoan
            set SoDuTK=SoDuTK+@sotien
            where MaTK=@tkden
        COMMIT TRAN
    end try
    begin catch
        declare @loi nvarchar(100)
        set @loi=N'Lỗi:'+ Error_message()
        raiserror (@loi,16,1)
        ROLLBACK TRAN
        return
    end catch
end
```

Quản lý các trạng thái Transaction

- **Active :** Trạng thái khởi đầu, giao dịch giữ trong trạng thái này trong khi nó đang thực hiện
- **Partially committed:** Sau khi lệnh cuối cùng được thực hiện.
- **Failed:** Sau khi phát hiện rằng sự thực hiện không thể tiếp tục được nữa.
- **Aborted:** Sau khi giao dịch đã bị "roll back" và CSDL đã phục hồi lại trạng thái của nó trước khi khởi động giao dịch.
- **Committed:** Sau khi thực hiện thành công giao dịch.

Quản lý các Transaction -trạng thái



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.

Mô hình Transaction trong SQL

- Cú Pháp:

BEGIN TRANSACTION

SQL Statements

COMMIT | ROLLBACK TRANSACTION

Mô hình Transaction trong SQL

Chúng ta có thể sử dụng TRY…CACTH hoặc IF cùng với TRANSACTION

```
BEGIN TRY
BEGIN TRAN
        -- Code for your transaction
COMMIT TRAN
END TRY
BEGIN CATCH
        -- output an error message
ROLLBACK TRAN
 END CATCH
```

Mô hình Transaction trong SQL

- Ví dụ 6.1: Giao tác dưới đây kết thúc do lệnh ROLLBACK TRANSACTION và mọi thay đổi về mặt dữ liệu mà giao tác đã thực hiện (UPDATE) đều không có tác dụng.

```
BEGIN TRANSACTION giaotac1
UPDATE monhoc SET sodvht=4 WHERE sodvht=3
UPDATE diemthi SET diemlan2=0 WHERE diemlan2 IS NULL
ROLLBACK TRANSACTION giaotac1
```

Mô hình Transaction trong SQL

- Giao tác dưới đây kết thúc bởi lệnh COMMIT và thực hiện thành công việc cập nhật dữ liệu trên các bảng MONHOC và DIEMTHI.

```
BEGIN TRANSACTION giaotac2
UPDATE monhoc SET sodvht=4 WHERE sodvht=3
UPDATE diemthi SET diemlan2=0 WHERE diemlan2 IS NULL
COMMIT TRANSACTION giaotac2
```

Mô hình Transaction trong SQL

```
declare @tranname varchar(20)
select @tranname ='MyTran'
Begin tran @tranname
        use Northwind
        delete from [Order Details] where
        OrderID=10248
RollBack transaction @tranname
select * from [Order Details] where OrderID=10248
Go
```

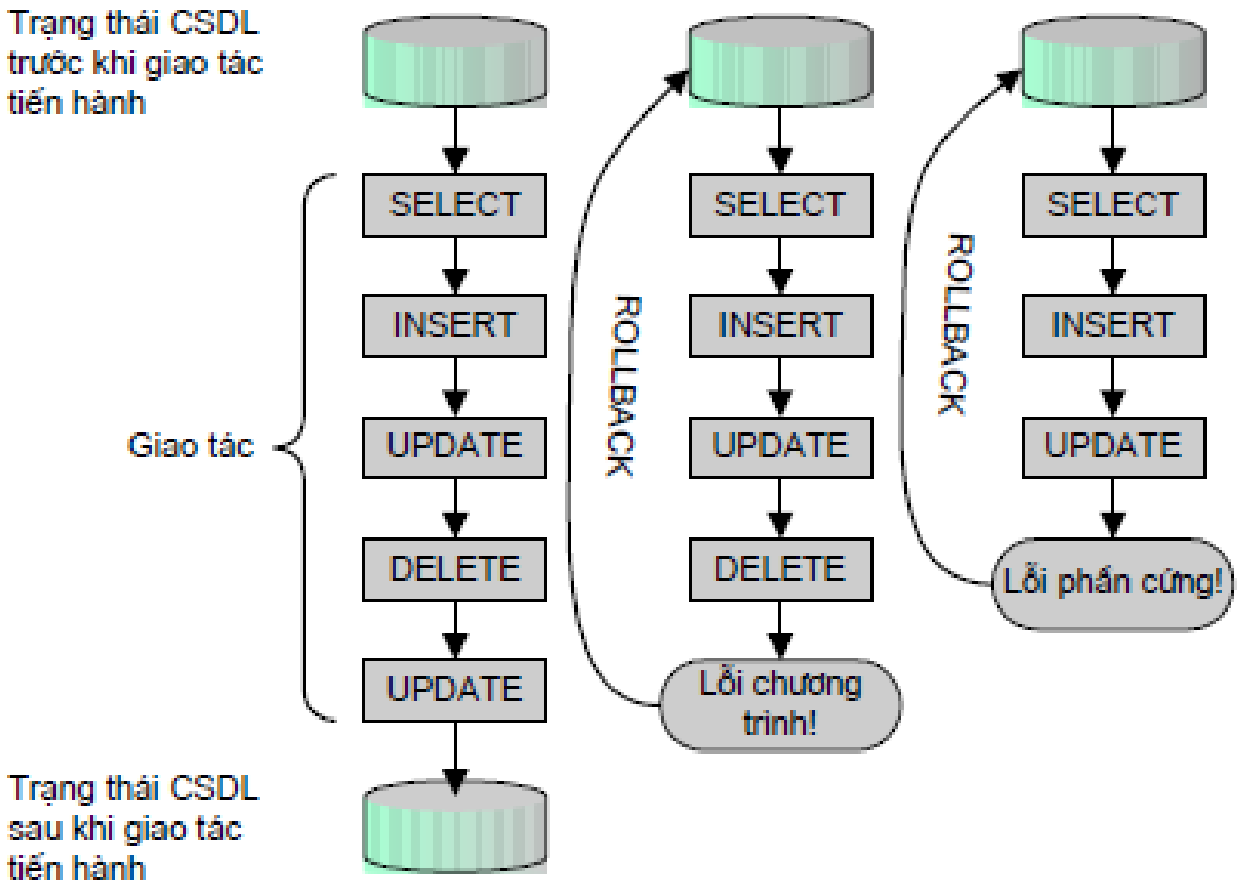
Mô hình Transaction trong SQL

```
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
```

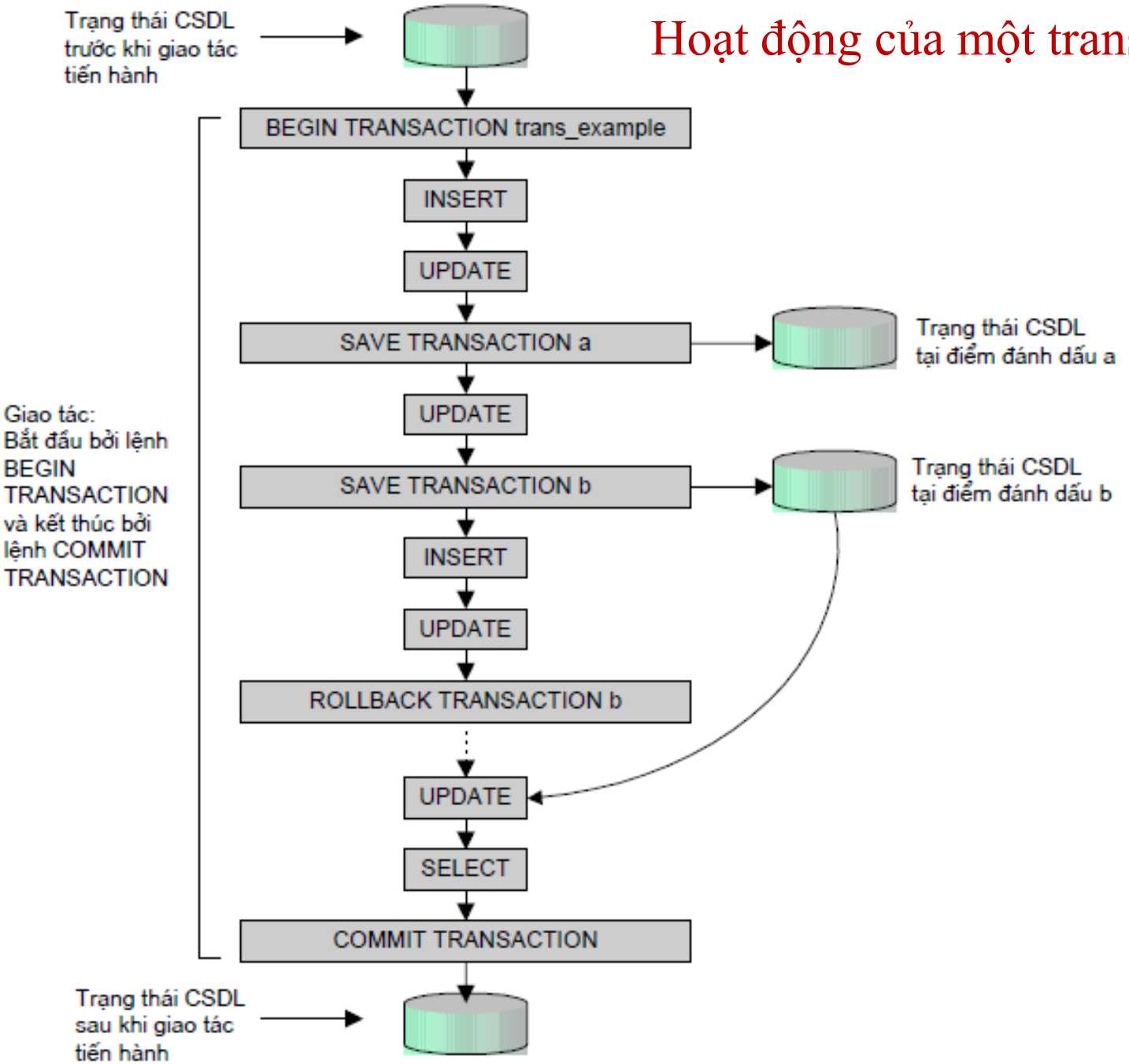
Mô hình Transaction trong SQL

```
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
```

Quản lý các Transaction -trạng thái



Hoạt động của một transaction



Transaction

- Ví dụ về rollback và commit transaction

```
Begin tran
--Thao tác 1
--Thao tác 2
Save tran nhan_1
--Thao tác 3
--Thao tác 4
Rollback tran nhan_1
--Thao tác 5
--Thao tác 6
Commit tran / Rollback tran
```

Trong ví dụ này, khi chạy đến dòng lệnh "Rollback tran nhan_1", các thao tác 3 và sẽ bị hủy bỏ. Trong khi ấy các thao tác 1 và 2 thì không bị hủy bỏ. Sau đó SQL Server sẽ vẫn tiếp tục làm nốt các thao tác 5 và 6.

Nếu câu lệnh cuối là "commit tran" thì các thao tác 1, 2, 5 và 6 sẽ được lưu bền vững vào cơ sở dữ liệu. Ngược lại nếu câu lệnh cuối là "rollback tran" thì các thao tác này sẽ bị hủy bỏ.

Using SAVE TRAN

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
```

Using SAVE TRAN

```
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 been
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)
```

Using SAVE TRAN

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
        begin catch
                rollback tran
                raiserror ('Transaction Error',16,1)
                return
        end catch
```

Các loại Transaction

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

Explicit Transaction_Transaction tường minh

- Là một Transaction phải định nghĩa bắt đầu một Transaction (Begin Transaction) và kết thúc một transaction(Commit Transaction)

```
BEGIN TRAN[SACTION]
    -- T-SQL statements
    -- @@TRANCOUNT = 1
    IF <some error condition>
        ROLL BACK
        ELSE
        COMMIT
        -- @@TRANCOUNT = 0
```

- **Tạo điểm lưu** (SAVE POINT)

```
save  tran tên_điểm_lưu
```

- Hủy những gì sau điểm lưu nếu **rollback**

Explicit Transaction_Transaction tường minh

- **begin tran t1**
  - lệnh | khối_lệnh
  - **save tran s1**
    - lệnh | khối_lệnh
  - **rollback tran s1** => chưa chấm dứt t1 lệnh | khối_lệnh
- **commit tran t1**

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

*=> **Rollback tran s1** chỉ hủy bỏ kết quả sau lệnh **save tran s1** và tiếp tục làm tiếp (giao tác t1 vẫn còn).*

Implicit Transactions(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.
- Bạn không cần bắt đầu một transaction, bạn chỉ cần Commit hay Rolback mỗi transaction.
- Chế độ Implicit transaction phát sinh một chuỗi các Transaction liên tục.
- **Thiết lập thông số** :

SET IMPLICIT_TRANSACTIONS ON|OFF

Implicit Transactions(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**

Autocommit Transaction

- **Au*tocommit* mode**: mô hình quản lý transaction mặc định của SQL Server.
- Một giao dịch được committed nếu thực hiện thành công hay trả ngược về lại ban đầu (roll back) nếu gặp lỗi.
- Lệnh **BEGIN TRANSACTION** vượt quyền autocommit mặc định.

Distributed Transactions

- Là một loại Explicip Transaction nhưng giao tác của nó liên quan nhiều Server. Sự quản lý phải được kết hợp giữa các nhà 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.

TRANSACTION LOG

- Dùng để theo vết tất cả các giao dịch
- 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 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 (tt)

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.)

Công dụng của Transation

- Phục hồi các Transaction đặc biệt: Khi môt Application đưa ra lệnh ROLL BACK hay SQL nhận ra một lỗi, thì bảng ghi log được dùng để Roll Back bất kỳ hiệu chỉnh 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.

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

Hàm XACT_STATE

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

1	**Yêu câu hiện thời có 1 transaction người dùng đang hoạt động có thể commit được**
0	Không có transaction nào đang hoạt động
-1	Yêu câu 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 câu không thể "write" được chon 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 uncommitable
                state' +'Rolling back transaction'
                Rollback tran
        END
END CATCH
```

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
```

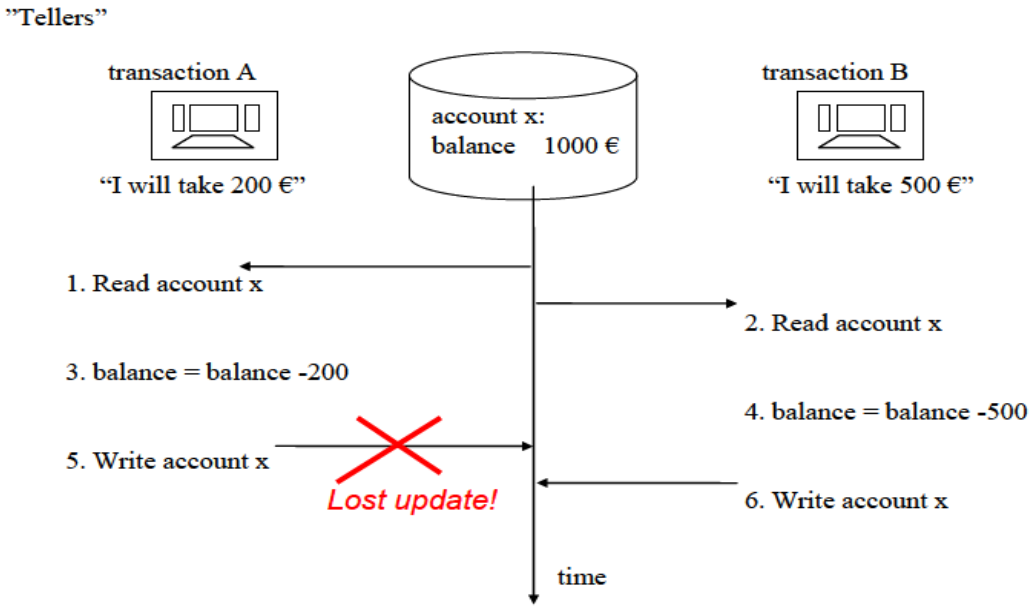
Điều khiển đồng thời

- 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 updated: Mất dữ liệu khi cập nhật
  - Dirty Read: Đọc dữ liệu rác
  - Unrepeatable Read: Không thể đọc lại
  - Phantom Read: Đọc dữ liệu ma.

Điều khiển đồng thời (tt)

Lost Updates (tổn thất cập nhật):

- Tình trạng này xảy ra khi có nhiều hơn một giao tác cùng thực hiện đọc và cập nhật trên 1 đơn vị dữ liệu. Khi đó, tác dụng của giao tác cập nhật thực hiện sau sẽ đè lên tác dụng của thao tác cập nhật trước.



Mất dữ liệu khi cập nhật - Lost updated

- Xét ví dụ:

- P_1 và P_2 xử lý đồng thời:

	P_1	P_2
t_1	Read(A) A=20	
t_2		Read(A) A=20
t_3	A=A-5 A=15	
t_4		A=A+10 A=30
t_5	Write(A) A=15	
t_6		Write(A) A=30
t_7	Read(A) A=30	

Điều khiển đồng thời (tt)

- Lost Updates

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	6
2	T1	Mark = mark +0.5	
3	T1	Ghi mark	6.5
4	T2	Đọc mark	6.5
5	T2	Mark = mark - 3	
6	T2	Ghi mark	3.5

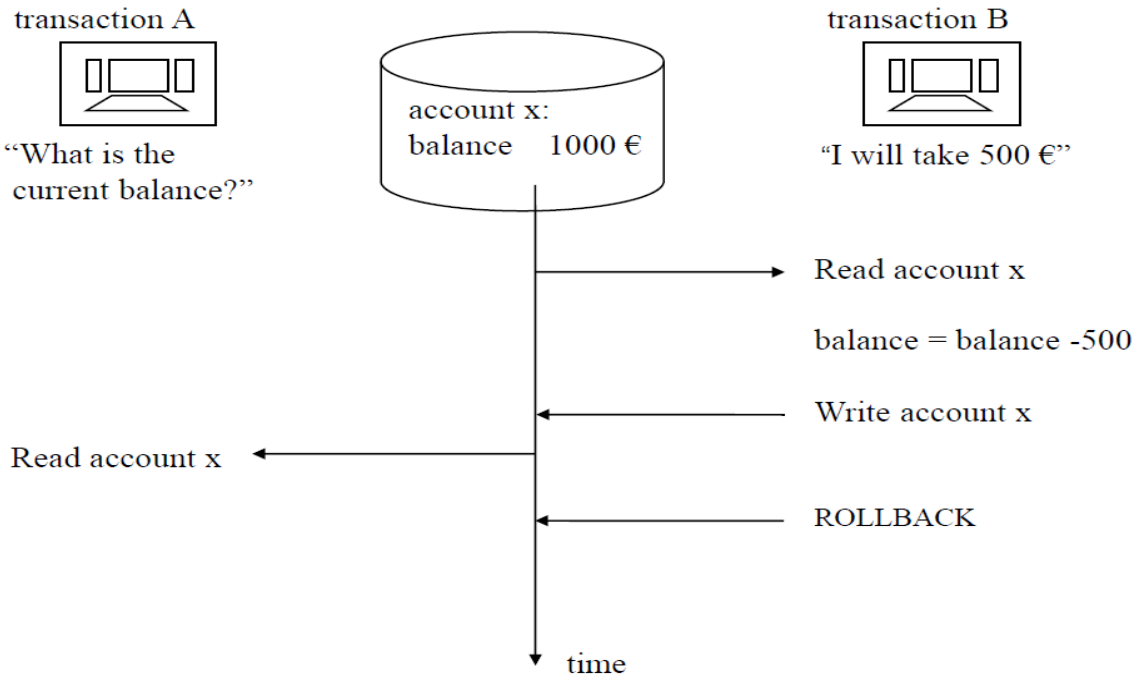
Điều khiển đồng thời (tt)

- Lost Updates

Time	Transaction	step	Stored valued
1	T1	Đọc mark	6
2	T2	Đọc mark	6
3	T1	Mark = mark + 0.5	
4	T2	Mark = mark – 3	
5	T1	Ghi mark	6.5
6	T2	Ghi mark	3.0

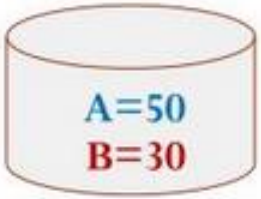
Điều khiển đồng thời (tt)

- Dirty Read (đọc dữ liệu sai): Xảy ra khi một giao tác thực hiện đọc trên một đơn vị dữ liệu mà đơn vị dữ liệu này đang bị cập nhật bởi một giao tác khác nhưng việc cập nhật chưa được xác nhận đã hoàn tất.



Đọc dữ liệu rác - Dirtty Read

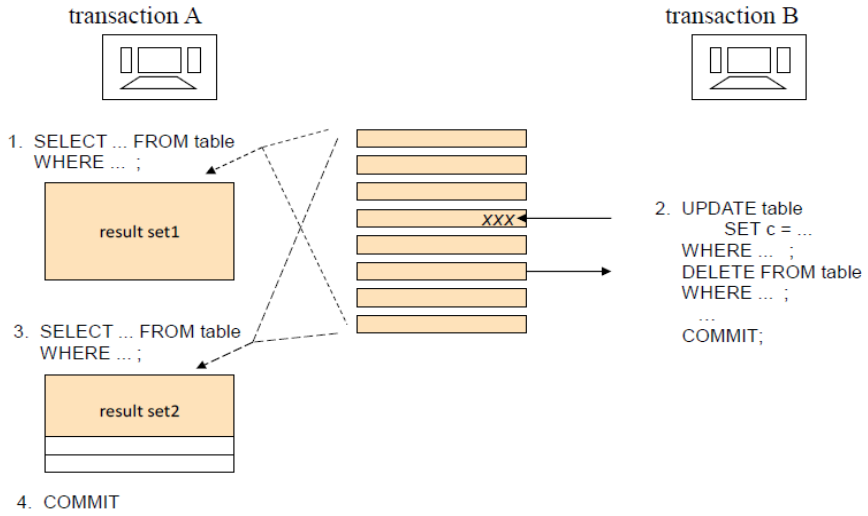
- P_1 và P_2 xử lý đồng thời:



	P_1	P_2
t_1	Read(A) A=50	
t_2		Read(B) B=30
t_3		B=B+10 B=40
t_4		Write(B) B=40
t_5	Read(B) B=40	
t_6	C=A+B C=90	
t_7	Print(C) C=90	
t_8		Rollback

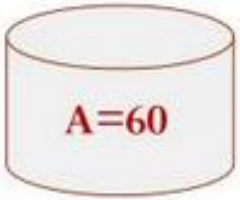
Điều khiển đồng thời (tt)

- Nonrepeatable Read: Tình trạng này xảy ra khi một giao tác A vừa thực hiện xong thao tác đọc trên một đơn vị dữ liệu thì giao tác khác B lại thay đổi (ghi) trên đơn vị dữ liệu này. Điều này làm cho lần đọc sau đó của A không còn nhìn thấy dữ liệu ban đầu nữa.



Không thể đọc lại - Unrepeatable Read

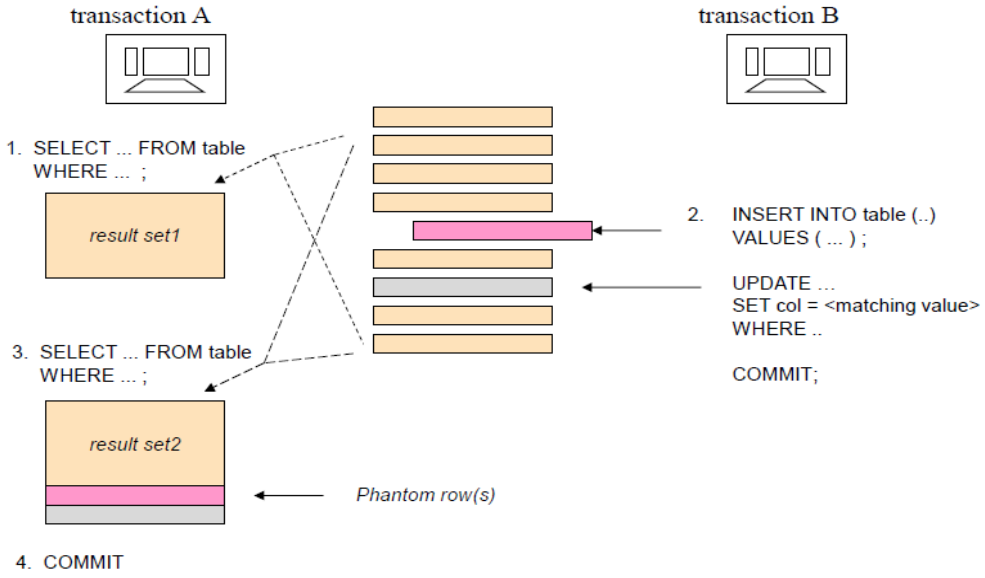
- P_1 và P_2 xử lý đồng thời:



	P_1		P_2	
t_1	Read(A)	A=50		
t_2	Print(A)	A=50		
t_3			Read(A)	A=50
t_4			A=A+10	A=60
t_5			Write(A)	A=60
t_6	Read(A)	A=60		

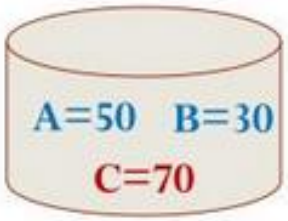
Điều khiển đồng thời (tt)

- Phantom Reads (đọc các mẫu tin ma): Là tình trạng mà một giao tác đang thao tác trên một tập dữ liệu nhưng giao tác khác lại chèn thêm hoặc xóa đi các dòng dữ liệu vào tập dữ liệu mà giao tác kia quan tâm.



Đọc dữ liệu ma – Phantom Read

- P_1 và P_2 xử lý đồng thời:



	P_1	P_2
t_1	Read(>40) A=50	
t_2		C=70
t_3		Write(C) C=70
t_4	Read(>40) A=50 C=70	

Điều khiển đồng thời (tt)

Uncommited data: mối quan hệ 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 và có thể bị thay đổi bởi giao tác đang thực hiện việc cập nhật.

Điều khiển đồng thời (tt)

- Uncommited data

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	6
2	T1	Mark = mark +0.5	
3	T1	Ghi mark	6.5
4	T1	Rollback	
5	T2	Đọc mark	6.0
6	T2	Mark = mark - 3	
7	T2	Ghi mark	3.0

Điều khiển đồng thời (tt)

- Uncommited data

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

Điều khiển đồng thời (tt)

Inconsistent retrievals: phân tích 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. Phân tích không nhất quán tương tự như mối 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 (tt)

- Inconsistent retrievals

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

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

Điều khiển đồng thời (tt)

- 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 chất của thao tác trong giao tác

- **Tính tương thích:**
  - Hai thao tác Oi, Oj là tương thích với nhau nếu kết quả của việc thực hiện đồng thời Oi,Oj cũng giống như kết quả của việc thực hiện tuần tự Oi, Oj hay Oj, Oi

	T_1	T_2	
O_{11}	Read A$\rightarrow a_1$ $a_1 + 1 \rightarrow a_1$ Print a_1	Read A$\rightarrow a_2$ a_2*2 $\rightarrow a_2$ Print a_2	O_{21}
O_{12}	Read A$\rightarrow a_1$ $a_1 + 5 \rightarrow a_1$ Write $a_1 \rightarrow$ A	Read A$\rightarrow a_2$ a_2*2 $\rightarrow a_2$ Write $a_2 \rightarrow$ A	O_{22}
O_{13}	Read A$\rightarrow a_1$ $a_1 + 2 \rightarrow a_1$ Write $a_1 \rightarrow$ A	Read A$\rightarrow a_2$ $a_2 + 7 \rightarrow a_2$ Write $a_2 \rightarrow$ A	O_{23}
O_{14}	Read B$\rightarrow b_1$ $b_1 + 1 \rightarrow b_1$ Write $b_1 \rightarrow$ B		

O_{11} và O_{21} là tương thích

O_{12} và O_{22} không tương thích

O_{13} và O_{23} không tương thích

O_{14} tương thích với các thao tác còn lại

Tính chất của thao tác trong giao tác

- **Tính khả hoán vị:**
  - Hai thao tác O_i và O_j (O_i thuộc T_i, O_j thuộc T_j) là khả hoán vị nếu kết quả thực hiện O_i ,O_j hay O_j, O_i là như nhau

	T_1	T_2	
O_{11}	Read A→ a_1 $a_1 + 1 \rightarrow a_1$ Print a_1	Read A→ a_2 $a_2 * 2 \rightarrow a_2$ Print a_2	O_{21}
O_{12}	Read A→ a_1 $a_1 + 5 \rightarrow a_1$ Write a_1→ A	Read A→ a_2 $a_2 * 2 \rightarrow a_2$ Write a_2→ A	O_{22}
O_{13}	Read A→ a_1 $a_1 + 2 \rightarrow a_1$ Write a_1→ A	Read A→ a_2 $a_2 + 7 \rightarrow a_2$ Write a_2→ A	O_{23}
O_{14}	Read B→ b_1 $b_1 + 1 \rightarrow b_1$ Write b_1→ B		

O_{11} và O_{21} là khả hoán vị

O_{12} và O_{22} không khả hoán vị

O_{13} và O_{23} là không khả hoán vị

O_{14} khả hoán vị với các thao tác còn lại

Tính chất của thao tác trong giao tác

- **Nhận xét:**
  - Các thao tác truy xuất các đơn vị dữ liệu khác nhau là tương thích và khả hoán vị
  - Các thao tác truy xuất trên cùng đơn vị dữ liệu:
    - Read – Read → khả hoán vị
    - Write – Write → không có tính khả hoán vị
    - Read – Write → không có tính khả hoán vị
    - Write – Read → không có tính khả hoán vị
  - **Hai thao tác không khả hoán vị thì gọi là xung đột**

Cơ chế khóa - Lock

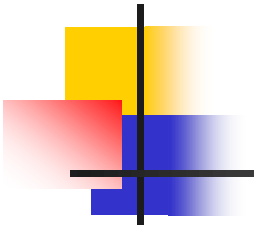
- Một giao dịch P trước khi muốn thao tác (Read/Write) trên một đơn vị dữ liệu A phải phát ra một yêu cầu xin khóa: Lock (A).
- Nếu yêu cầu được chấp nhận thì giao dịch được phép thao tác trên dữ liệu A
- Sau khi thao tác xong thì giao dịch P phải phát ra lệnh giải phóng A: unlock (A)

Cơ chế khóa - Lock

- Ví dụ:

	T_1	T_2
t_1	Read(A), A=50	
t_2		Read(A), A=50
t_3	A=A-30 A=20	
t_4		A=A+30 A=80
t_5	Write(A) A=20	
t_6		Write(A), Unlock(A) A=80
t_7	Read(A), Unlock(A) A=80	

	T_1	T_2
t_1	Lock(A),Read(A)	
t_2	A=A-30	
t_3	Write(A)	
t_4	Read(A), Unlock(A)	
t_5		Lock(A), Read(A)
t_6		A=A+30
t_7		Write(A), Unlock(A)



Types of Locks

- Basic locks
  - Shared (LS)
  - Exclusive (LX)
- Special situation locks
  - Intent
  - Update
  - Schema
  - Bulk update

Types of Locks

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

Khóa đọc và khóa ghi

- **Read lock = Shared lock (Slock)**
  - Giao dịch giữ **Slock** được phép **ĐỌC** dữ liệu, nhưng không được phép ghi.
  - ***Nhiều giao dịch có thể đồng thời giữ nhiều Slock*** trên cùng một đơn vị dữ liệu.
- **Write lock = Exclusive lock (Xlock)**
  - Giao dịch giữ **Xlock** được phép **GHI+ ĐỌC** dữ liệu
  - ***Tại một thời điểm chỉ có tối đa một giao dịch được quyền giữ Xlock*** trên một đơn vị dữ liệu.
  - Không thể thiết lập Slock trên đơn vị dữ liệu đang có Xlock

Khóa dự định ghi – Update lock

- Ulock được sử dụng khi đọc dữ liệu với dự định ghi lại dữ liệu này.
- Ulock là trung gian giữa Slock và Xlock
- Khi thực hiện thao tác ghi lên dữ liệu thì bắt buộc Ulock phải tự động chuyển thành Ulock
- Giao dịch giữ **Ulock** được phép **ĐỌC và GHI** dữ liệu.
- Tại một thời điểm chỉ có một giao tác được quyền giữa Ulock trên một đơn vị dữ liệu.
- Có thể thiết lập Slock trên đơn vị dữ liệu có Ulock

Khóa trực tiếp trong câu lệnh

- **Cú pháp**

```
SELECT ...
FROM table1 WITH (lock1 [, lock2,…] )
WHERE ...


DELETE FROM table1 WITH (lock1 [, lock2, …])
WHERE ...


UPDATE table1 WITH  (lock1 [, lock2, …])
SET ...
WHERE ...
```

Các mức cô lập của giao dịch

- **Mục đích**:
  - Tự động đặt khóa cho các thao tác đọc trong kết nối đế dữ liệu hiện hành
- **SQL hỗ trợ các mức cô lập**
  - Read Uncommitted
  - Read Committed (The default)
  - Repeatable Read
  - Serializable

Các mức cô lập của SQL Server DB Engi

1. Read uncommitted: các lệnh có thể đọc các hàng bị chỉnh 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. Reapeated 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.

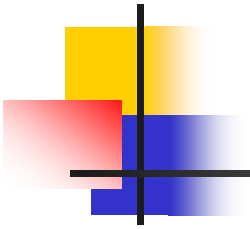
Lệnh thay đổi mức cô lập

- 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
```



Read uncommited

- Đặc điểm
  - Đọc dữ liệu: không cần thiết lập Slock
  - Ghi dữ liệu: SQL Server tự động thiết lập Xlock trên đơn vị dữ liệu được ghi. ***Xlock được giữ cho đến hết giao dịch***

Read uncommited

- Khi transaction thực hiện ở mức này, các truy vấn vẫn có thể truy cập 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**

Read uncommitted và Lost Updated

KHACHHANG

MaKH	TenKH	DiaChi	DienThoai
KH001	XYZ	Đà Nẵng	0511.3246135
KH002	Cửa hàng Hoàng Gia	Quảng Nam	0510.6333444
KH003	Nguyễn Lan Anh	Huế	0988.148248
KH004	Công ty TNHH An Phước	Đà Nẵng	0511.6987789
KH005	Huỳnh Ngọc Trung	Quảng Nam	0905.888555
KH006	Cửa hàng Trung Tín	Đà Nẵng	*NULL*

Xlock

Xlock

T1

```
BEGIN TRAN
 SET TRANSACTION ISOLATION LEVEL
 READ UNCOMMITTED
 UPDATE KhachHang
 SET TenKH= 'ABC'
 WHERE MaKH= 'KH001'

 WAITFOR DELAY '00:00:05'

 SELECT TenKH
 FROM KhachHang
 WHERE MaKH= 'KH001'
COMMIT TRAN
```

T2

```
BEGIN TRAN
 SET TRANSACTION ISOLATION LEVEL
 READ UNCOMMITTED
 UPDATE KhachHang
 SET TenKH= 'XYZ'
 WHERE MaKH= 'KH001'
COMMIT TRAN
```

Read uncommitted và Lost Updated

KHACHHANG

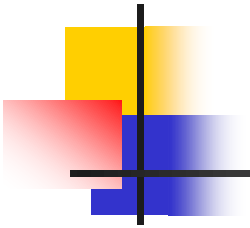
MaKH	TenKH	DiaChi	DienThoai
KH001	ABC	Đà Nẵng	0511.3246135
KH002	ửa hàng Hoàng Gia	Quảng Nam	0510.6333444
KH003	Nguyễn Lan Anh	Huế	0988.148248
KH004	Công ty TNHH An Phước	Đà Nẵng	0511.6987789
KH005	Huỳnh Ngọc Trung	Quảng Nam	0905.888555
KH006	Cửa hàng Trung Tín	Đà Nẵng	*NULL*

Xlock

T1	T2
BEGIN TRAN ***SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED*** UPDATE KhachHang SET TenKH= 'ABC' WHERE MaKH= 'KH001' WAITFOR DELAY '00:00:05' ROLLBACK TRAN	BEGIN TRAN ***SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED*** SELECT TenKH FROM KhachHang WHERE MaKH= 'KH001' COMMIT TRAN

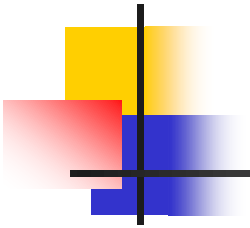
Read uncommited

- **Ưu điểm:**
  - Giải quyết vấn đề Lost Updated
  - Không cần thiết lập Slock, không cản trở giao dịch khác giữa Xlock.
- **Hạn chế:**
  - Có khả năng xảy ra 3 vấn đề của truy xuất đồng thời: Dirty Read, Unrepeatable Read, Phantom



Read commited

- **Đặc điểm**
  - **Đọc dữ liệu**: SQL tự động thiết lập Slock trên đơn vị dữ liệu được đọc, Slock được giải phóng ngay khi đọc xong.
  - **Ghi dữ liệu**: SQL tự động thiết lập Xlock trên đơn vị dữ liệu được ghi, Xlock được giữ cho đến khi kết thúc giao dịch.

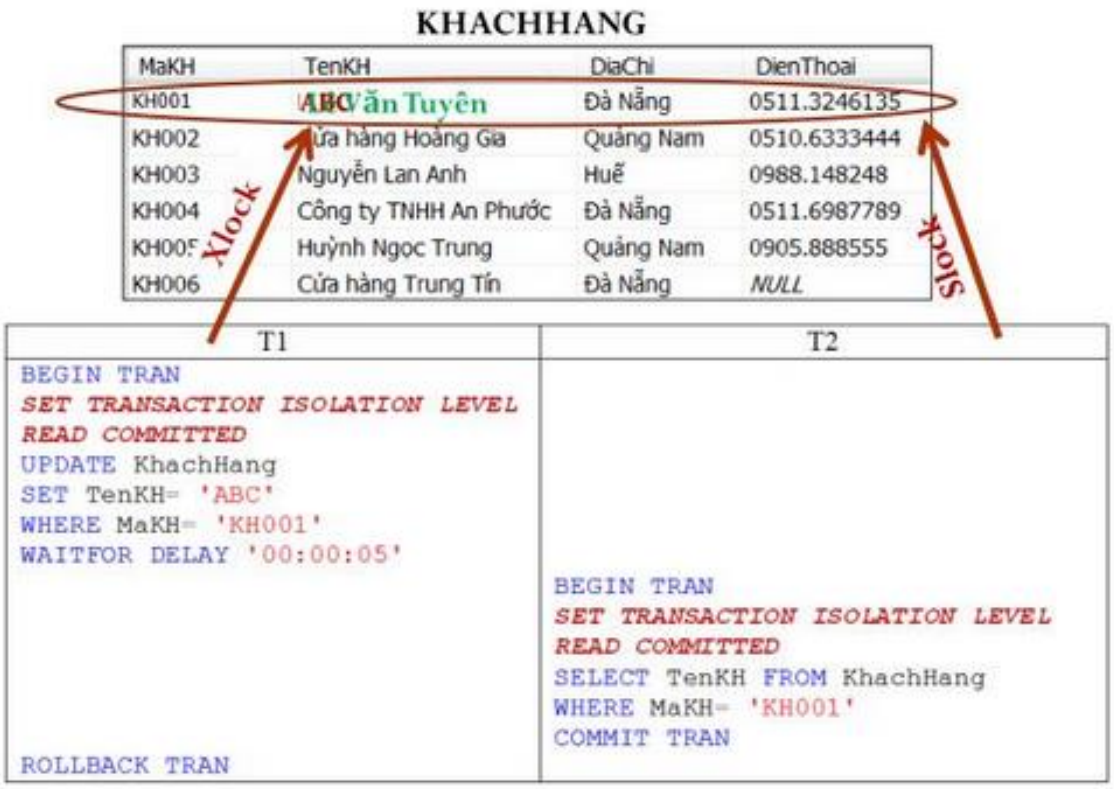


Read commited

- Đây là mức isolation mặc định. Transaction sẽ không đọc được dữ liệu đang được cập nhật mà phải đợi đến khi việc cập nhật thực hiện xong.
- Tránh được dirty read như ở mức trên

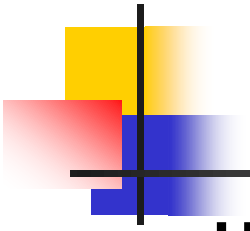
Read commited và Dirty read

- Ví dụ:



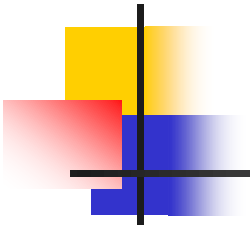
Read commited và Unrepeatable read

- Ví dụ



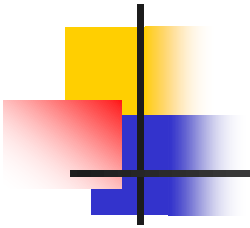
Read commited

- **Ưu điểm**
  - Giải quyết được vấn đề Dirty read, Lost Updated
  - Slock được giải phóng ngay, không cản trở nhiều đến thao tác ghi dữ liệu của các giao tác khác
- **Hạn chế**
  - Chưa giải quyết được vấn đề Unrepeatable read, Phantom



Repeatable read

- **Đặc điểm**
  - **Đọc dữ liệu**: SQL tự động thiết lập Slock trên đơn vị dữ liệu được đọc và giữ Slock cho đến khi kết thúc giao dịch
  - **Ghi dữ liệu**: SQL tự động thiết lập Xlock trên đơn vị dữ liệu được ghi, Xlock được giữ đến khi kết thúc giao dịch

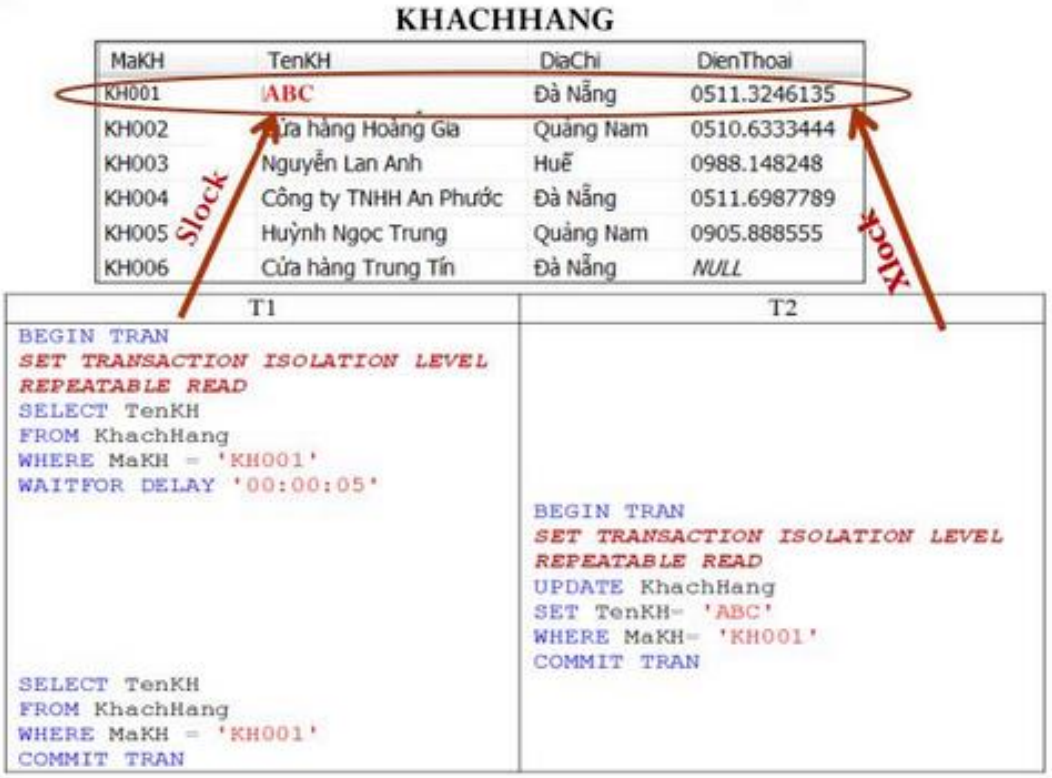


Repeatable read

- 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

Repeatable read và vấn đề Unrepeatable read

- Ví dụ:



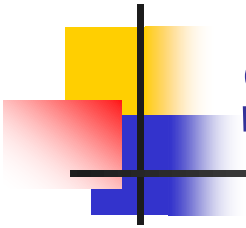
Repeatable read và vấn đề Phantom read

- Ví dụ

T1	T2
BEGIN TRAN SET TRANSACTION ISOLATION LEVEL REPEATABLE READ SELECT * FROM HangHoa WHERE SLCon = 100 WAITFOR DELAY '00:00:05' SELECT * FROM HangHoa WHERE SLCon = 100 COMMIT TRAN	BEGIN TRAN SET TRANSACTION ISOLATION LEVEL REPEATABLE READ INSERT INTO HangHoa VALUES ('IP','Ipad','Cái',100,10000000) COMMIT TRAN

Repeatable read

- **Ưu điểm**
  - Giải quyết được 3 vấn đề: Lost Updated, Dirty read, và Unrepeatable read
- **Hạn chế**
  - Chưa giải quyết được vấn đề Phantom, do vẫn cho phép insert những dòng dữ liệu thỏa điều kiện thiết lập của Slock
  - Slock được giữ cho đến khi kết thúc giao dịch, cản trở việc truy cập dữ liệu của các giao dịch khác



Serializable

- **Đặc điểm**
  - **Đọc dữ liệu**: SQL tự động thiết lập Slock trên đơn vị dữ liệu đọc và giữ cho đến khi kết thúc giao dịch
  - Không cho phép thêm những dòng dữ liệu thỏa mãn điều kiện thiết lập trên Slock
  - **Ghi dữ liệu**: SQL tự động thiết lập Xlock trên đơn vị dữ liệu được ghi và Xlock được giữ cho đến khi giao dịch kết thức

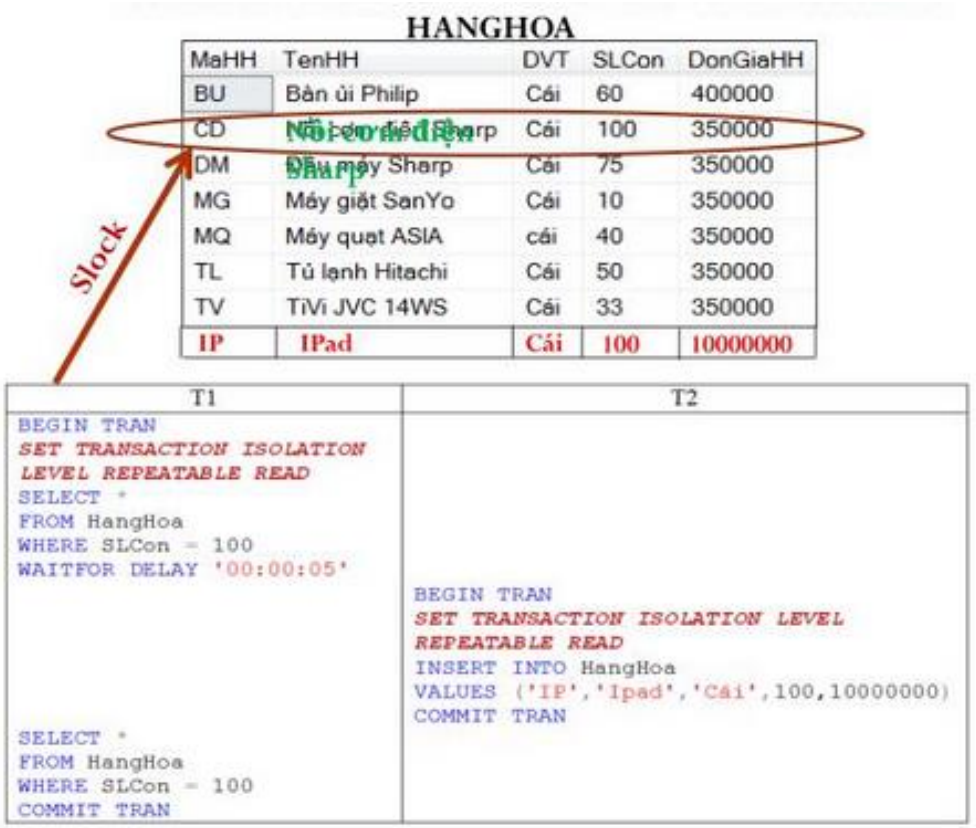
Serializable

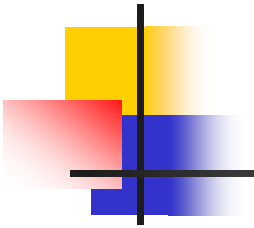
- Mức isolation này tăng thêm một cấp nữa và khóa toàn bộ dải các bản ghi có thể bị ảnh hưởng bởi một transaction khác, dù là UPDATE/DELETE bản ghi đã có hay INSERT bản ghi mới. Nếu bạn thay cửa sổ 1 bằng đoạn code

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRAN
```

Serializable và vấn đề Phantom

- Ví dụ





Serializable

- **Ưu điểm**
  - Giải quyết được 4 vấn đề: Lost Updated, Dirty Read, Unrepeatable read, và Phantom
- **Hạn chế**
  - Slock được giữ cho đến hết giao dịch, cản trở việc cập nhật dữ liệu của các giao dịch khác
  - Không cho phép insert những dòng dữ liệu thỏa mãn điều kiện thiết lập Slock, cản trở việc thêm mới dữ liệu của các giao dịch khác

Lịch biểu (SCHEDULE)

- Lịch biểu (SCHEDULE): Là một dãy (có thứ tự) các thao tác của một tập các giao dịch mà trong đó thứ tự của các thao tác trong mỗi giao dịch được bảo toàn.

Ví dụ:

T1
Read(A)
A = A – 10
Write(A)
Read(B)
B = B + 10
Write(B)

T2
Read(B)
B = B – 20
Write(B)
Read(C)
C = C + 20
Write(C)

T1	T2
Read(A)	
	Read(B)
A = A – 10	
	B = B – 20
Write(A)	
	Write(B)
Read(B)	
	Read(C)
B = B + 10	
	C = C + 20
Write(B)	
	Write(C)

Lịch biểu (SCHEDULE)

- Lịch biểu tuần tự (Serial Schedule): Là lịch biểu mà trong mỗi giao dịch các thao tác được thực hiện kế tiếp nhau, không có thao tác của giao dịch khác xen vào (Thực hiện lần lượt, hết giao dịch này đến giao dịch khác)

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

T1
Read(A)
A = A – 10
Write(A)
Read(B)
B = B + 10
Write(B)

T2
Read(B)
B = B – 20
Write(B)
Read(C)
C = C + 20
Write(C)

T1	T2
Read(A)	
A = A – 10	
Write(A)	
Read(B)	
B = B + 10	
Write(B)	
	Read(B)
	B = B – 20
	Write(B)
	Read(C)
	C = C + 20
	Write(C)

Lịch biểu (SCHEDULE)

- Lịch biểu không tuần tự (Nonserial Schedule): Là lịch biểu mà các thao tác trong các giao dịch được đan xen vào nhau.

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

T1	T2
Read(A)	Read(B)
A = A – 10	B = B – 20
Write(A)	Write(B)
Read(B)	Read(C)
B = B + 10	C = C + 20
Write(B)	Write(C)

T1	T2
Read(A)	
	Read(B)
A = A – 10	
	B = B – 20
Write(A)	
	Write(B)
Read(B)	
	Read(C)
B = B + 10	
	C = C + 20
Write(B)	
	Write(C)

Lịch biểu (SCHEDULE)

- Lịch biểu gọi là khả tuần tự (Serializable): nếu nó tương đương với một lịch biểu tuần tự.
- Tương đương theo nghĩa cho ra cùng một trạng thái CSDL sau khi kết thúc việc thực hiện lịch biểu.
- Lịch biểu bất khả tuần tự nếu nó không tương đương với một lịch biểu tuần tự.

T1	T2
Read(A)	
	Read(B)
A = A – 10	
	B = B – 20
Write(A)	
	Write(B)
Read(B)	
	Read(C)
B = B + 10	
	C = C + 20
Write(B)	
	Write(C)

Lịch biểu (SCHEDULE)

Tìm lịch biểu tương đương bằng phương pháp hoán vị các thao tác liền nhau của 2 giao tác nếu có thể

T1	T2	A	B	C
Read(A)		60	30	10
A = A-10				
Write(A)		50	30	10
Read(B)			30	
B = B+10				
Write(B)		50	40	10
	Read(B)		40	
	B = B-20			
	Write(B)	50	20	10
	Read(C)			10
	C = C+20			
	Write(C)	50	20	30

T1	T2	A	B	C
Read(A)		60	30	10
	Read(B)		30	
A = A-10				
	B = B-20			
Write(A)		50	30	10
	Write(B)	50	10	10
Read(B)			10	
	Read(C)			10
B = B+10				
	C = C+20			
Write(B)		50	20	10
	Write(C)	50	20	30

Lịch biểu (SCHEDULE)

- Ví dụ: Lịch biểu bất khả tuần tự

T1	T2	A	B	C
Read(A)		60	30	10
A = A-10				
Write(A)		50	30	10
Read(B)			30	
B = B+10				
Write(B)		50	40	10
	Read(B)		40	
	B = B-20			
	Write(B)	50	20	10
	Read(C)			10
	C = C+20			
	Write(C)	50	20	30

T1	T2	A	B	C
Read(A)		60	30	10
A = A-10				
	Read(B)		30	
Write(A)		50	30	10
	B = B-20			
Read(B)			30	
	Write(B)	50	10	10
B = B+10				
	Read(C)			10
Write(B)		50	40	10
	C = C+20			
	Write(C)	50	40	30

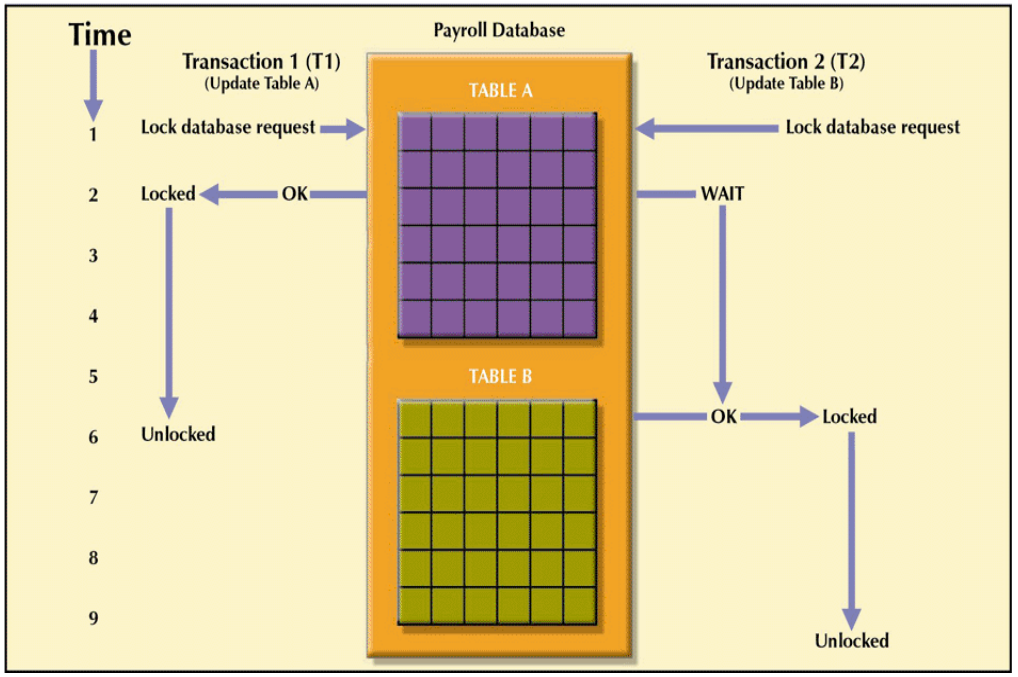
Phương pháp Locking

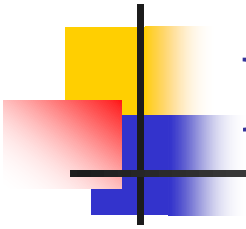
- Dùng để điều khiển đồng thời
  - 1 lock được cấp để sử dụng “độc quyền” 1 hạng mục dữ liệu trong transaction hiện hành.
  - 1 transaction được cấp lock trước khi truy cập dữ liệu; sau khi transaction hoàn tất, lock phải được giải phóng
  - Lock manager quản lý những thông tin về lock.

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 lock
  - Binary Locks
    - Có 2 trạng thái: locked (1) or 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 đụng độ dữ liệu.
    - Một exclusive lock 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 là 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 cho các hàng đọc được trong bảng kết nối
    - Một khóa exclusive cho các hàng được cập nhật trong bảng update.

Phương pháp Locking (tt)

- Các kiểu lock

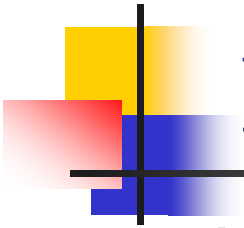
  - Shared Locks
    - Một shared lock tồn tại khi các transaction đồng thời đọc dữ liệu
    - Một shared lock không làm đụng độ dữ liệu khi các transaction đồng thời chỉ đọc dữ liệu
    - Một shared lock được gán khi transaction muốn đọc dữ liệu và dữ liệu đó không tồn tại exclusive lock.

Phương pháp Locking (tt)

- Các kiểu lock
  - Intent Lockss
    - DB Engine dùng khóa này để bảo vệ việc đặt khóa S hay X trên tài nguyên ở mức thấp hợn. 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 mà không cần phải tìm từng khóa trên mỗi trang hay mỗi hàng của bảng đó.



Phương pháp Locking (tt)

- Hai vấn đề với pp locking
  - Schedule của transaction không khả tuần tự
  - Có thể tạo ra deadlock
- Giải pháp
  - Khả tuần tự: two phase locking
  - Deadlock: phát hiện và ngăn chặn

Phương pháp Locking (tt)

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

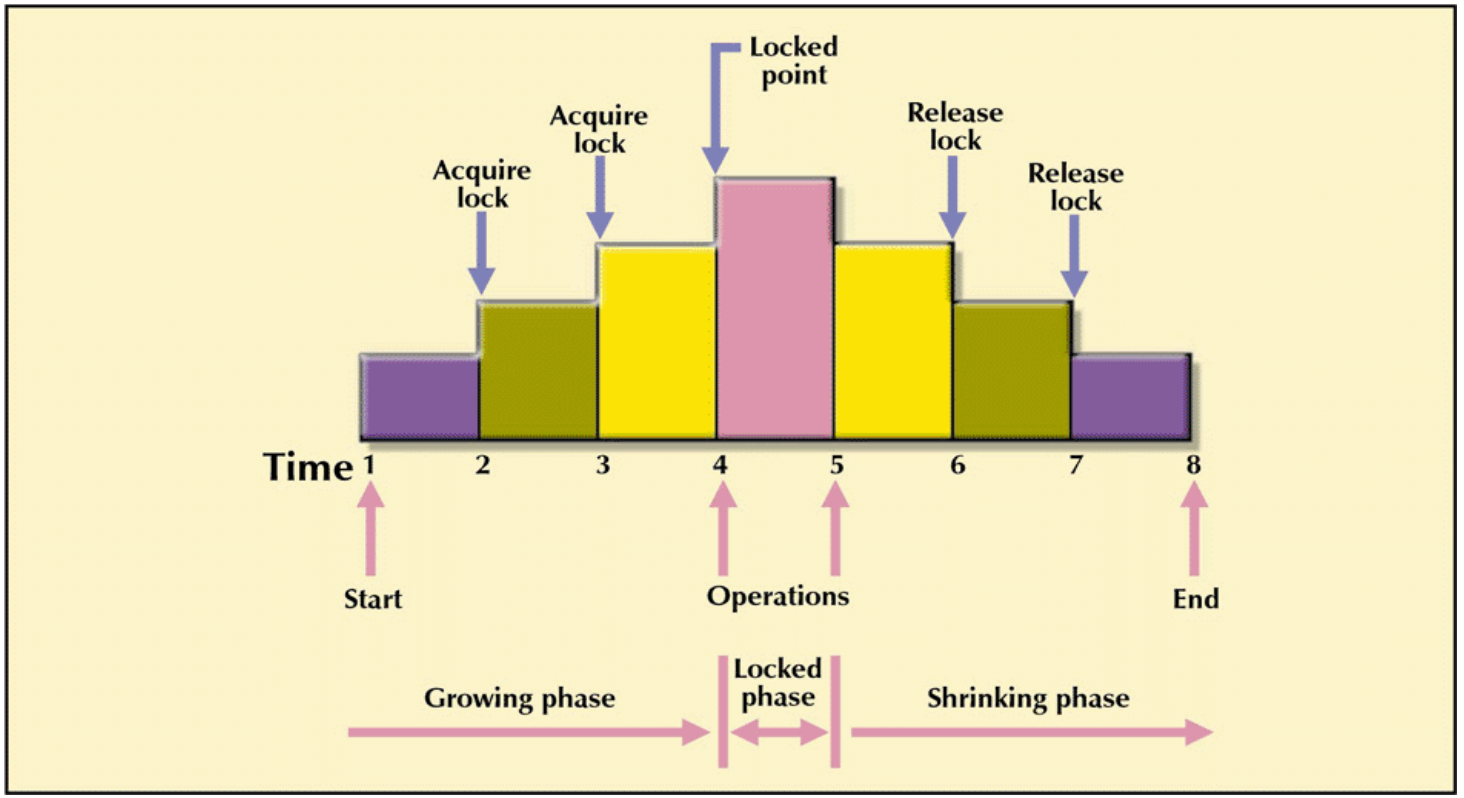
Phương pháp Locking (tt)

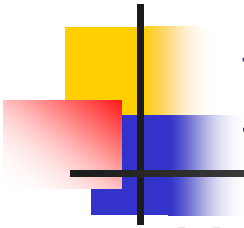
- Locking two-phase
  - Qui tắc cho giao thức Two-Phase Locking
    - Không có 2 transaction nào có khóa đụng độ
    - Trong cùng 1 transaction không có thao tác không khóa nào đi trước thao tác có khóa.
    - Không có dữ liệu nào bị ảnh hưởng cho đến khi tất cả các khóa lấy được.

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 mà 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 mà DBMS phải phát hiện thấy và phải giải quyết gỡ bỏ nghẽn khóa.
- Chỉ có 1 cách là hủy bỏ 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 = access data items X and Y
    - T2 = access data items Y and 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

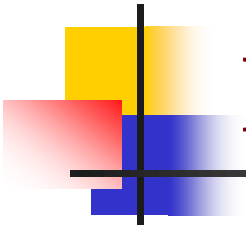
TIME	TRANSACTION	REPLY	LOCK STATUS		
0			Data X		Data Y
1	T1:LOCK(X)	OK	Unlocked		Unlocked
2	T2: LOCK(Y)	OK	Locked		Unlocked
3	T1:LOCK(Y)	WAIT	Locked		Locked
4	T2:LOCK(X)	WAIT	Locked	**Deadlock**	Locked
5	T1:LOCK(Y)	WAIT	Locked		Locked
6	T2:LOCK(X)	WAIT	Locked		Locked
7	T1:LOCK(Y)	WAIT	Locked		Locked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

Phương pháp Locking (tt)

- Deadlock: 3 kỹ thuật để điều khiển Deadlocks:
  - Chặn Deadlock: Một transaction sẽ bị từ bỏ nếu yêu cầu lock mới và 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 từ bỏ 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 (Multigranular locking)

- 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 share trên các hàng, và khóa intent share 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 combatibility

- Tương thích khóa dùng để kiểm soát nhiều transaction có 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 mode của khóa được yêu cầu tương thích với mode 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.

Sự tương thích khóa Lock combatibility

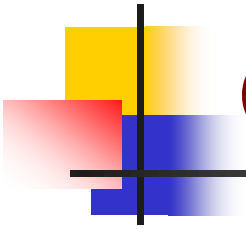
Ví dụ: Sự tương thích khóa Lock combatibility:

- Không có mode 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 Lock combatibility

Ví dụ: Sự tương thích khóa Lock combatibility:

	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.

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.