

Chương 6

Quản lý giao tác và khóa (Transactions & Lock)

Giáo trình & Tài liệu tham khảo:

- 1. Microsoft SQL Server 2008 R2 Unleashed**, Ray Rankins, Paul Bertucci, Chris Gallelli, Alex T. Silverstein, 2011, Pearson Education, Inc
- 2. MS SQL Server 2012 T-SQL fundamentals**, Tizik Ben-Gan
- 3. <https://docs.microsoft.com/>**

Khái niệm Transaction

- Transaction cho phép các tác vụ thao tác trên database (một hay nhiều câu lệnh SQL) được xử lý như **một nguyên tử**, nghĩa là : các lệnh trong một transaction hoặc tất cả được thực thi thành công, hoặc không lệnh nào được thực thi.
- Sử dụng transaction giúp duy trì tính toàn vẹn và nhất quán của database , đặc biệt **trong môi trường nhiều người dùng**.
- <https://learn.microsoft.com/en-us/archive/blogs/florinlazar/why-and-when-to-use-transactions>

Khái niệm Transaction

- Một transaction được
 - bắt đầu bằng **Begin Transaction**
 - kết thúc bằng **Commit** (khi tất cả các lệnh thực thi thành công) hoặc **Rollback** (không lệnh nào thực thi)

Sử dụng lệnh

```
1. Begin the transaction
2. Process database commands
3. Check for errors
   If error occurs
       Roll back the transaction
   Else
       Commit the transaction
```

BEGIN TRANSACTION
ROLLBACK TRANSACTION
COMMIT TRANSACTION

Khái niệm Transaction

- Lệnh **COMMIT TRANSACTION** đánh dấu sự kết thúc của một transaction thành công \Leftrightarrow SQL Server lưu trữ bền vững trong database tất cả thay đổi dữ liệu, và giải phóng các tài nguyên do transaction nắm giữ
- Lệnh **ROLLBACK TRANSACTION** đánh dấu sự kết thúc của một transaction KHÔNG thành công \Leftrightarrow SQL Server xóa tất cả các sửa đổi dữ liệu được thực hiện từ khi bắt đầu transaction hoặc tới điểm đã lưu, và giải phóng các tài nguyên do transaction nắm giữ
 - Thông thường **ROLLBACK TRAN** được dùng khi một lệnh bị lỗi hoặc có vi phạm logic nghiệp vụ

Khái niệm Transaction

- Ví dụ 1 : Trong SQL Server , mỗi câu lệnh T-SQL được xem như một **Auto-Commit** transaction, nghĩa là SQL Server sẽ **tự động** Commit khi câu lệnh thực thi thành công, và Rollback khi câu lệnh lỗi

```
Create table Customer (  
  CustomerID int primary key,  
  CustomerName varchar(50)  
)
```

Khái niệm Transaction

User thực thi lần lượt 2 lệnh insert

[SQL Server tự động Begin transaction]

INSERT INTO Customer VALUES (1, 'David')

[SQL Server tự động Commit transaction]

[SQL Server tự động Begin transaction]

INSERT INTO Customer VALUES (1, 'Maria')

[SQL Server tự động Rollback transaction]

Khái niệm Transaction

- Ví dụ 2 : chuyển tiền 100\$ từ tài khoản A sang tài khoản B gồm 2 thao tác độc lập về xử lý , nhưng liên quan về logic nghiệp vụ
 - Trừ tiền tài khoản A đi 100\$
 - Tăng tiền trong tài khoản B 100\$
- => Người lập trình muốn 2 thao tác này đặt trong một transaction để đảm bảo : hoặc cả hai được thực hiện thành công, hoặc cả hai không được thực hiện

Khái niệm Transaction

- Ví dụ 2 xét 2 table

```
create table taikhoanA (  
sotien money check (sotien>=0)  
)
```

```
create table taikhoanB (  
sotien money check (sotien>=0)  
)
```

```
Insert into taikhoanA values (4000)
```

```
Insert into taikhoanB values (0)
```


Khái niệm Transaction

- Ví dụ 2

Đặt trong 1 batch

GO

```
Declare @no money  
Set @no = 5000  
Update taikhoanA  
set sotien = sotien - @no  
Update taikhoanB  
set sotien = sotien + @no
```

GO

=>Nhận xét : không đảm bảo
logic nghiệp vụ

Đặt trong 1 transaction

Declare @no money

Set @no = 5000

BEGIN TRANSACTION ----

Update taikhoanA

set sotien = sotien - @no

Update taikhoanB

set sotien = sotien + @no

COMMIT TRANSACTION ----

=>Nhận xét : không đảm bảo
logic nghiệp vụ

Khái niệm Transaction

Đặt trong 1 transaction
(cách 1) **

=> *Nhận xét : đảm bảo logic
nghịệp vụ*

```
DECLARE @no money
```

```
SET @no = 5000
```

```
BEGIN TRANSACTION
```

```
UPDATE taikhoanA SET sotien = sotien - @no
```

```
IF(@@ERROR != 0)
```

```
begin
```

```
ROLLBACK TRANSACTION
```

```
RETURN
```

```
end
```

```
UPDATE taikhoanB SET sotien = sotien + @no
```

```
IF(@@ERROR != 0)
```

```
begin
```

```
ROLLBACK TRANSACTION
```

```
RETURN
```

```
end
```

```
COMMIT TRANSACTION
```

Khái niệm Transaction

Đặt trong 1 transaction
(cách 2) **

=> *Nhận xét : đảm bảo logic
nghịệp vụ*

```
DECLARE @no money
SET @no = 5000
BEGIN TRANSACTION
UPDATE taikhoanA SET sotien = sotien - @no
IF(@@ERROR != 0)
    ROLLBACK TRANSACTION
ELSE
    begin
        UPDATE taikhoanB SET sotien = sotien + @no
        IF(@@ERROR != 0)
            ROLLBACK TRANSACTION
        COMMIT TRANSACTION
    end
```

Ví dụ

----bài tập transaction:

--thực hiện cập nhật listprice của mặt hàng mã 780 trong product

--và lưu lại listprice cũ trong productlistpriceHistory

--yêu cầu : hoặc cả 2 lệnh thực thi thành công, hoặc rollback nếu

-- 1 trong 2 lệnh bị lỗi (cú pháp, runtime, vi phạm RB)

go

begin transaction

- -- lưu listprice vào biến oldprice
- -- update listprice trong product
- -- insert vào bảng history
- -- nếu cả 2 lệnh thực thi thành công thì kết thúc tran bằng commit
- -- ngược lại nếu 1 trong 2 lệnh có lỗi thì kết thúc tran bằng rollback
- -- dùng biến global @@error để phát hiện lệnh có lỗi cú pháp hay lỗi runtime

Khái niệm Transaction

- Ví dụ 3 : xét table

```
CREATE TABLE TestTable  
(name char(1))
```

--Thực hiện lần lượt các transaction, và kiểm tra kết quả sau mỗi transaction

```
BEGIN TRANSACTION  
INSERT INTO TestTable VALUES('A');  
INSERT INTO TestTable VALUES('B');  
GO  
ROLLBACK TRANSACTION
```

Khái niệm Transaction

BEGIN TRANSACTION

INSERT INTO TestTable VALUES('A');

INSERT INTO TestTable VALUES('B');

GO

COMMIT TRANSACTION

BEGIN TRANSACTION

INSERT INTO TestTable VALUES('CD');

INSERT INTO TestTable VALUES('E');

GO

COMMIT TRANSACTION

Ba loại transaction

- **AutoCommit transaction**

- Mỗi câu lệnh T-SQL là một transaction và SQL Server tự động commit khi lệnh thực thi thành công , hoặc tự động rollback khi lệnh bị lỗi
- là default mode

- **Explicit transaction**

- Cung cấp cách tạo transaction theo logic người lập trình mong muốn (*a user-defined transaction*)
- Sử dụng các lệnh BEGIN TRAN, COMMIT/ROLLBACK TRAN/WORK
- Một số lệnh không thể cùng một transaction với lệnh khác

Ba loại transaction

- **Implicit transaction**

- Kết thúc một transaction là bắt đầu một transaction mới (*không cần lệnh Begin Transaction*)
- Phải kết thúc tường minh một transaction bằng lệnh COMMIT/ROLLBACK TRAN/WORK
- Là a user-defined transaction giống **Explicit transaction**
- Sử dụng lệnh để bật/tắt Implicit mode
SET IMPLICIT_TRANSACTIONS ON | OFF

Bốn đặc tính của transaction

ACID

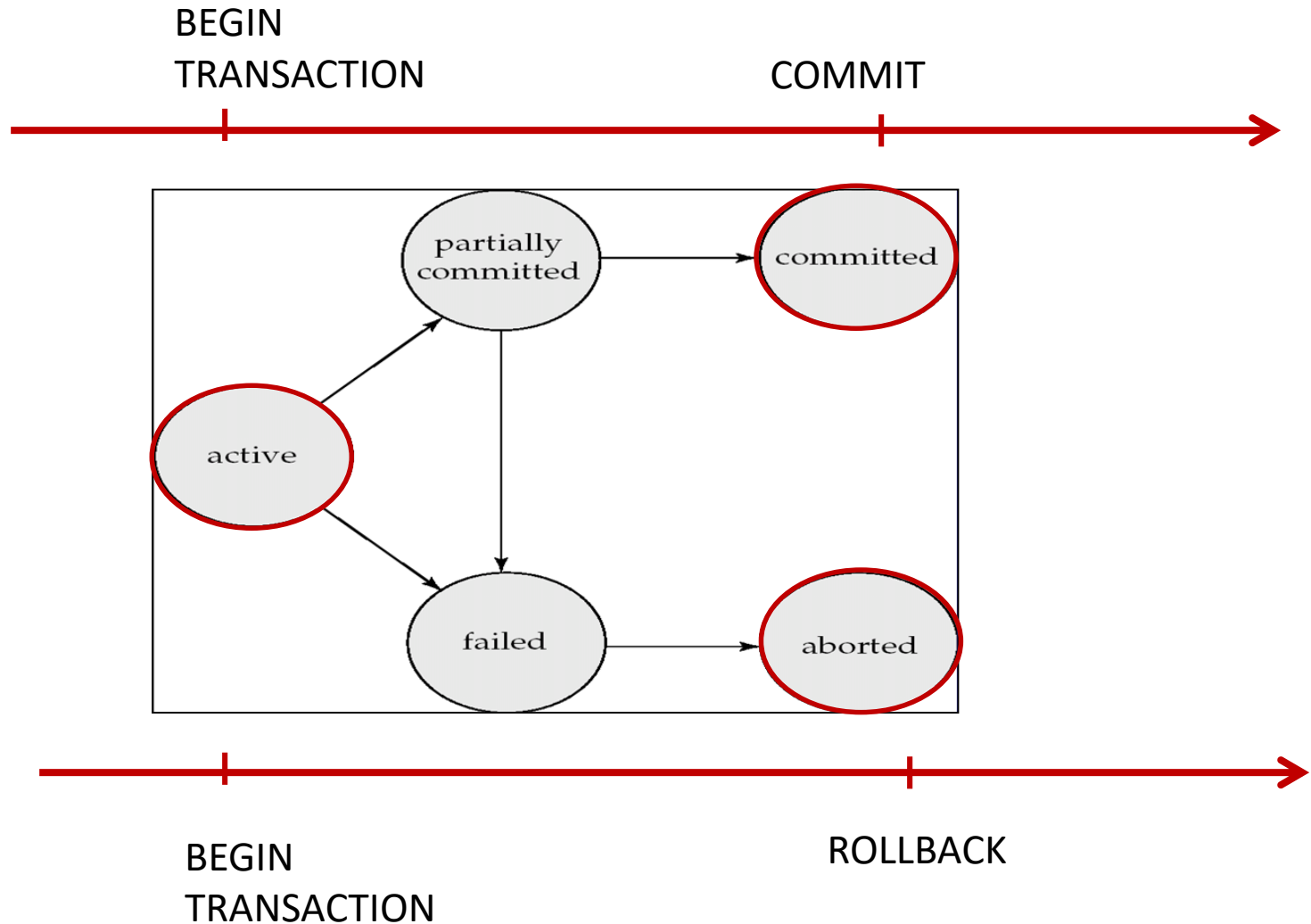
- Transaction giúp duy trì tính nhất quán và toàn vẹn của dữ liệu trong database
- Mỗi transaction có 4 đặc tính
 - Atomicity (nguyên tử)
 - Consistency (nhất quán)
 - Isolation (cô lập)
 - Durability (bền vững)
- Việc đảm bảo 4 đặc tính này được phân chia giữa người lập trình và SQL Server
 - Người lập trình đảm bảo các thao tác trong transaction là Atomicity và Consistency
 - SQL Server đảm bảo Isolation và Durability

Bốn đặc tính của transaction

ACID

- **Atomicity** (nguyên tử) : Transaction thành công nếu tất cả công việc trong transaction thực thi thành công, các thay đổi data trong database được ghi bền vững trong database. Ngược lại, nếu bất kỳ công việc nào trong transaction bị lỗi thì toàn bộ transaction lỗi và thay đổi data nếu có trong database sẽ được rollback
- **Consistency** (nhất quán) : transaction không ảnh hưởng (vi phạm) những ràng buộc dữ liệu trong database
- **Isolation** (cô lập) : Mỗi transaction là độc lập, không ảnh hưởng tới xử lý của transaction khác
- **Durability** (bền vững) : Sau khi transaction hoàn thành, mọi thay đổi được thực hiện bởi transaction được lưu bền vững trong database

Các trạng thái của transaction



User-defined Transactions

Save Transaction

- Sử dụng **SAVE TRANSACTION**
 - Đánh dấu một vị trí (savepoint) trong transaction.
Cho phép rollback một phần của transaction tới vị trí đã đánh dấu , thay vì rollback toàn bộ transaction

- Ví dụ 4 : xét table

```
CREATE TABLE TestTable (  
  ID INT NOT NULL PRIMARY KEY,  
  Value INT NOT NULL  
)
```

User-defined Transactions

Save Transaction

- Ví dụ 4

```
TRUNCATE TABLE TestTable
```

```
BEGIN TRANSACTION
```

```
INSERT INTO TestTable( ID, Value )
```

```
VALUES ( 1, N'10' )
```

```
-- this will create a savepoint after the first INSERT
```

```
SAVE TRANSACTION FirstInsert
```

```
INSERT INTO TestTable( ID, Value )
```

```
VALUES ( 2, N'20' )
```

```
-- this will rollback to the savepoint right after the first INSERT was done
```

```
ROLLBACK TRANSACTION FirstInsert
```

```
-- this will commit the transaction leaving just the first INSERT
```

```
COMMIT
```

User-defined Transactions

Save Transaction

- Ví dụ 4 (tiếp)

```
TRUNCATE TABLE TestTable
```

```
BEGIN TRANSACTION
```

```
INSERT INTO TestTable( ID, Value )
```

```
VALUES ( 1, N'10')
```

```
SAVE TRANSACTION FirstInsert
```

```
INSERT INTO TestTable( ID, Value )
```

```
VALUES ( 2, N'20')
```

```
ROLLBACK TRANSACTION FirstInsert
```

```
INSERT INTO TestTable( ID, Value )
```

```
VALUES ( 3, N'30')
```

```
COMMIT
```

User-defined Transactions

@@TRANCOUNT

- Sử dụng @@TRANCOUNT
 - Trả về số lượng transaction đang mở trong phiên làm việc hiện tại
 - @@TRANCOUNT sẽ giảm đi 1 khi kết thúc một transaction bằng lệnh commit hay rollback
- Ví dụ 5 : xét table

```
CREATE TABLE TestTable (  
  ID INT NOT NULL PRIMARY KEY,  
  Value INT NOT NULL  
)
```

User-defined Transactions

@@TRANCOUNT

- Ví dụ 5

```
TRUNCATE TABLE TestTable
```

```
BEGIN TRANSACTION
```

```
INSERT INTO TestTable( ID, Value )
```

```
VALUES ( 1, N'10')
```

```
GO
```

```
select @@TRANCOUNT
```

```
BEGIN TRANSACTION
```

```
INSERT INTO TestTable( ID, Value )
```

```
VALUES ( 2, N'20')
```

```
GO
```

```
select @@TRANCOUNT
```

```
COMMIT
```

```
select @@TRANCOUNT
```

```
COMMIT
```

```
select @@TRANCOUNT
```


User-defined Transactions

SET XACT_ABORT

- Thiết lập SET XACT_ABORT ON
 - Nếu một câu lệnh trong transaction phát sinh **lỗi run-time** => SQL Server **tự động** rollback transaction tại thời điểm phát sinh lỗi
 - Compile errors như syntax errors, không chịu ảnh hưởng bởi SET XACT_ABORT
- Cú pháp
SET XACT_ABORT { ON | OFF }

User-defined Transactions

SET XACT_ABORT

- Ví dụ 6

Xét table

Create table Customer (
CustomerID int primary key,
CustomerName varchar(50))

SET XACT_ABORT OFF;


Begin tran

```
INSERT INTO Customer VALUES (1, 'David')  
INSERT INTO Customer VALUES (1, 'David')  
INSERT INTO Customer VALUES (2, 'Maria')
```

Commit tran

```
Select * from Customer
```

Rollback toàn bộ
transaction do lỗi run-time
của câu lệnh thứ 2



SET XACT_ABORT ON;

Begin tran

```
INSERT INTO Customer VALUES (3, 'Kenvin')  
INSERT INTO Customer VALUES (3, 'Kenvin')  
INSERT INTO Customer VALUES (4, 'Tom')
```

Commit tran

```
Select * from Customer
```

User-defined Transactions

A nested transaction

- Là transaction nằm bên trong transaction khác
- Ví dụ 7 : xét table

```
CREATE TABLE Table1 (  
  ID INT DEFAULT 1,  
  Value INT DEFAULT 10 )
```

User-defined Transactions

A nested transaction

Nhận xét 2 đoạn lệnh ?

```
truncate table table1
```

```
BEGIN TRAN Tran1
```

```
GO
```

```
BEGIN TRAN Nested
```

```
GO
```

```
INSERT INTO TABLE1 DEFAULT Values
```

```
GO 10
```

```
COMMIT TRAN Nested
```

```
SELECT * FROM Table1
```

```
COMMIT TRAN Tran1
```

```
SELECT * FROM Table1
```

```
truncate table table1
```

```
BEGIN TRAN Tran1
```

```
GO
```

```
BEGIN TRAN Nested
```

```
GO
```

```
INSERT INTO TABLE1 DEFAULT Values
```

```
GO 10
```

```
COMMIT TRAN Nested
```

```
SELECT * FROM Table1
```

```
ROLLBACK TRAN Tran1
```

```
SELECT * FROM Table1
```

User-defined Transactions

Lưu ý

Certain commands cannot be specified within a user-defined transaction, primarily because they cannot be effectively rolled back in the event of a failure.

- ALTER DATABASE
ALTER FULLTEXT CATALOG
ALTER FULLTEXT INDEX
BACKUP DATABASE
BACKUP LOG
CREATE DATABASE
CREATE FULLTEXT CATALOG
CREATE FULLTEXT INDEX
DROP DATABASE
DROP FULLTEXT CATALOG
DROP FULLTEXT INDEX
RESTORE DATABASE
RECONFIGURE
RESTORE LOG
UPDATE STATISTICS

Ví dụ

- Tạo một transaction giảm giá (Listprice) mặt hàng xe đạp có mã 780 trong bảng Product xuống 10% **nếu** tổng trị giá tồn kho của mặt hàng này sau khi giảm giá không thấp hơn 60% so với tổng trị giá tồn kho với đơn giá ban đầu

```
Begin transaction
declare @giacu money , @giamoi money
select @giacu = ListPrice , @giamoi = ListPrice*0.9
from Production.Product
where productID = 780
```

```
declare @tongtrigiaTonkhoGiacu money, @tongtrigiaTonkhoGiamoi money
select @tongtrigiaTonkhoGiacu = sum(Quantity*@giacu) , @tongtrigiaTonkhoGiamoi = sum(Quantity*@giamoi)
from Production.ProductInventory
where ProductID = 780
```

```
if (@tongtrigiaTonkhoGiamoi >= @tongtrigiaTonkhoGiacu*0.6 )
begin
update Production.Product
set ListPrice = ListPrice*0.9
where productID = 780
print N'đã cập nhật giá giảm 10%'
commit
end
else
begin
print N'không giảm giá được'
commit
end
```

LOCKING

- **Quản lý truy suất đồng thời** (*Concurrent data access*)
 - SQL Server cho phép nhiều user cùng truy suất data tại một thời điểm => Cần cơ chế để ngăn ngừa những tác động ngược chiều khi nhiều user cố gắng truy suất cùng một tài nguyên mà user khác đang sử dụng ở cùng thời điểm
- => nhằm đảm bảo tính toàn vẹn của transactions và duy trì sự nhất quán dữ liệu của database

LOCKING

4 vấn đề

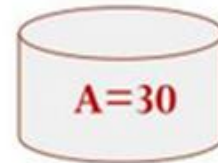
- **Các vấn đề** có thể xảy ra khi truy suất đồng thời
 - Lost Updated (mất dữ liệu khi cập nhật)
 - Dirty Read (Đọc dữ liệu rác)
 - Unrepeatable Read (không thể đọc lại)
 - Phantom Read (đọc dữ liệu ma)

LOCKING

4 vấn đề

- **Lost Updated** : khi hai hay nhiều transaction thao tác cập nhật trên cùng một dữ liệu. Thao tác cập nhật sau cùng của một transaction chép đè lên cập nhật trước đó của một transaction khác. Kết quả là mất data.

- P_1 và P_2 xử lý đồng thời:



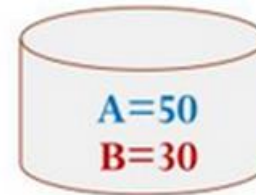
	P_1	P_2
t_1	Read(A) $A=20$	
t_2		Read(A) $A=20$
t_3	$A=A-5$ $A=15$	
t_4		$A=A+10$ $A=30$
t_5	Write(A) $A=15$	
t_6		Write(A) $A=30$
t_7	Read(A) $A=30$	

LOCKING

4 vấn đề

- **Dirty Read** : Khi một transaction đọc dữ liệu đang được cập nhật bởi một transaction khác, nhưng việc cập nhật chưa được xác nhận là hoàn tất

P_1 và P_2 xử lý đồng thời:



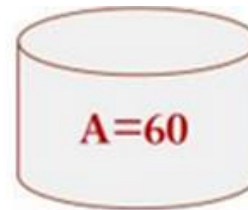
	P_1	P_2
t_1	Read(A) $A=50$	
t_2		Read(B) $B=30$
t_3		$B=B+10$ $B=40$
t_4		Write(B) $B=40$
t_5	Read(B) $B=40$	
t_6	$C=A+B$ $C=90$	
t_7	Print(C) $C=90$	
t_8		Rollback

LOCKING

4 vấn đề

- **Unrepeatable Read** : một transaction thay đổi dữ liệu mà một transaction khác đang đọc, khiến cho các lần đọc cùng một dữ liệu cho kết quả khác nhau

- P_1 và P_2 xử lý đồng thời:



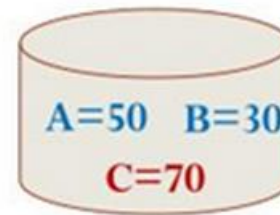
	P_1	P_2
t_1	Read(A) A=50	
t_2	Print(A) A=50	
t_3		Read(A) A=50
t_4		A=A+10 A=60
t_5		Write(A) A=60
t_6	Read(A) A=60	

LOCKING

4 vấn đề

- **Phantom Read** : một transaction thực hiện 2 truy vấn giống hệt nhau trên một tập dữ liệu nhưng kết quả khác nhau, do một transaction khác thực hiện *thêm/xóa* dòng liên quan đến tập dữ liệu của câu truy vấn

• 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	

LOCKING

Isolation Levels

- Isolation Levels
 - Cho phép một transaction lựa chọn các mức độ bảo vệ khỏi sự ảnh hưởng của transaction khác trên dữ liệu mà nó đang thao tác
 - Sử dụng lệnh với các mức độ bảo vệ

```
SET TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SNAPSHOT
| SERIALIZABLE
}
```

LOCKING

Isolation Levels

- Ví dụ : thiết lập Isolation Level cho connection hiện hành

```
USE AdventureWorks2012;
```

```
GO
```

```
---thiết lập mức độ bảo vệ "REPEATABLE READ"
```

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

```
GO
```

```
BEGIN TRANSACTION;
```

```
GO
```

```
SELECT *
```

```
    FROM HumanResources.EmployeePayHistory;
```

```
GO
```

```
SELECT *
```

```
    FROM HumanResources.Department;
```

```
GO
```

```
COMMIT TRANSACTION;
```

```
GO
```

LOCKING

4 Isolation Levels

- READ UNCOMMITTED
 - Cho phép một transaction có thể đọc dữ liệu đang được cập nhật bởi transaction khác trước khi transaction hoàn tất
 - Mức bảo vệ thấp nhất , nguy cơ xảy ra *dirty read*
- READ COMMITTED
 - Không cho phép một transaction đọc dữ liệu mà transaction khác đang update chưa hoàn tất; nhưng không bảo vệ transaction đọc dữ liệu (transaction khác có thể làm thay đổi dữ liệu đang được đọc)
 - Nguy cơ xảy ra *nonrepeatable reads* , hay *phantom data*
 - Là thiết lập default của SQL Server

LOCKING

4 Isolation Levels

- REPEATABLE READ

- Không cho phép một transaction update dữ liệu khi một transaction khác đang đọc
- Chỉ bảo vệ data đang tồn tại, không ngăn được việc thêm dữ liệu mới => nguy cơ xảy ra *phantom rows*

- SERIALIZABLE

- Không cho phép một transaction update dữ liệu khi một transaction khác đang đọc, và ngăn việc thêm hay xóa dòng khỏi tập dữ liệu
- Là mức bảo vệ cao nhất , tránh được 4 vấn đề

LOCKING

Isolation Levels

- Nhận xét : Khi các transaction đọc cùng một row, rồi sau đó cập nhật row đã đọc thì nên chạy ở Isolation level là **repeatable read** để ngăn chặn Lost Updated

Isolation Level	Dirty Read	Non Repeatable Read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot	No	No	No
Serializable	No	No	No

LOCKING

Giới thiệu

- The SQL Server Database Engine sử dụng 2 cơ chế để đảm bảo tính toàn vẹn của các transaction
 - **Locking**
 - **Row Versioning**
- **Locking**
 - Mỗi transaction sẽ yêu cầu lock trên tài nguyên mà nó đang truy suất (rows, pages, tables)
 - Lock sẽ khóa (block) các transaction khác chỉh sửa tài nguyên mà gây ảnh hưởng đến transaction yêu cầu lock
 - Transaction sử dụng lock sẽ giải phóng lock khi nó không còn truy suất tài nguyên

LOCKING

Giới thiệu

1. Transaction T muốn truy suất (reading/modifying) data A
=> Transaction T yêu cầu một lock trên data A
2. Transacion T thực hiện thao tác trên data A , và nắm giữ lock
Thời gian nắm giữ lock phụ thuộc vào thiết lập Isolation Level hiện hành
3. Transaction T hoàn tất (commit / rollback) => giải phóng lock

LOCKING

Quá trình cấp Lock

Cấp lock được thực hiện bởi Lock Manager :

- Khi xử lý một truy vấn , Query processor của SQL Server Database Engine sẽ xác định tài nguyên cần truy suất
- Query processor xác định loại lock cần để bảo vệ tài nguyên dựa trên dạng câu lệnh SQL và thiết lập Transaction Isolation Level hiện hành
- Query processor sẽ gửi yêu cầu về Locks tới Lock Manager
- Lock Manager sẽ cấp locks nếu không có xung đột về locks đang nắm giữ bởi các transaction khác

LOCKING

Lock types

- ✓ **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

LOCKING

Ví dụ

- Xét table

```
create table Accounts (  
  AccountID int NOT NULL PRIMARY KEY,  
  balance int NOT NULL  
  CONSTRAINT unloanable_account CHECK (balance >= 0) )
```

----refresh data

```
truncate table Accounts
```

```
select * from Accounts
```

```
INSERT INTO Accounts (AccountID,balance) VALUES (101,1000);
```

```
select * from Accounts
```

LOCKING

Ví dụ 0

SET TRANSACTION ISOLATION LEVEL **READ UNCOMMITTED**

go

---- In A client window

Begin tran

update Accounts

set balance = balance - 200

go

update Accounts

set balance = balance - 500

select * from Accounts

Commit

Non Lost Updated

---- In B client window

Begin tran

update Accounts

set balance = balance +2000

Commit

LOCKING

Ví dụ 1

SET TRANSACTION ISOLATION LEVEL **READ UNCOMMITTED**
go

Dirty Read

---- In A client window

Begin tran

update Accounts set balance = 200

go

Rollback

select * from Accounts

---- In B client window

Begin tran

select * from Accounts

Commit

LOCKING

Ví dụ 2

SET TRANSACTION ISOLATION LEVEL **READ COMMITTED**
go

---- In A client window

Begin tran

update Accounts set balance = 200
go

select * from Accounts

Commit --- or Rollback



---- In B client window

Begin tran

select * from Accounts
Commit

LOCKING

Ví dụ 3

SET TRANSACTION ISOLATION LEVEL **READ COMMITTED**

go

UnRepeatable Read

---- In A client window

Begin tran

select * from Accounts

go

select * from Accounts

Commit

select * from Accounts

---- In B client window

Begin tran

update Accounts set balance = 500

select 'After', * from Accounts

Commit

LOCKING

Ví dụ 4

SET TRANSACTION ISOLATION LEVEL **REPEATABLE READ**
go

---- In A client window
Begin tran
select * from Accounts
go

select * from Accounts
Commit
select * from Accounts



---- In B client window

Begin tran
update Accounts set balance = 500
select 'After', * from Accounts
Commit

LOCKING

Ví dụ 5

SET TRANSACTION ISOLATION LEVEL **REPEATABLE READ**

go

---- *In A client window*

Begin tran

update Accounts set
balance = 200

go

select * from Accounts

Commit

select * from Accounts

---- *In B client window*

Begin tran

select 'Before', * from Accounts

update Accounts set balance = 500

select 'After', * from Accounts

Commit

LOCKING

Ví dụ 6

SET TRANSACTION ISOLATION LEVEL **REPEATABLE READ**

go

---- *In A client window*

Begin tran

INSERT INTO Accounts
VALUES (201,2000)

Commit

Phantom Read

---- *In B client window*

Begin tran

select 'Before', * from Accounts

select 'After', * from Accounts

Commit

LOCKING

Ví dụ 7

SET TRANSACTION ISOLATION LEVEL **SERIALIZABLE**
go



---- In A client window

Begin tran
INSERT INTO Accounts
VALUES (201,2000)
Commit



---- In B client window

Begin tran
select 'Before', * from Accounts



select 'After', * from Accounts
Commit



TÓM TẮT

Bài tập

- Giả sử có 2 giao dịch chuyển tiền từ tài khoản 101 và 202 như sau:
 - Client A chuyển 100\$ từ tài khoản 101 sang 202
 - Client B chuyển 200\$ từ tài khoản 202 sang 101

Viết các lệnh tương ứng ở 2 client để kiểm soát các giao dịch xảy ra đúng ?

Bài tập

Dữ liệu ban đầu

```
create table Accounts (  
    AccountID int NOT NULL PRIMARY KEY,  
    balance int NOT NULL  
    CONSTRAINT unloanable_account CHECK (balance >= 0) )
```

```
INSERT INTO Accounts VALUES (101,1000);
```

```
INSERT INTO Accounts VALUES (202,2000);
```