Chương 5: Transaction

Khái niệm về giao dịch - 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

Các thuộc tính của Transaction - ACID

- **Atomic (nguyên tố)**: nếu một lệnh nào đó trong transaction bị fail thì dữ liệu trở lại trạng thái ban đầu.
- **Consistency (Nhất quán)**: giao dịch không phá vỡ trạng thái nhất quán của CSDL.
- **Isolation (cô lập)**: giao dịch không ảnh hưởng hoặc bị ảnh hưởng bởi giao dịch khác.
- **Durability (Bền)**: 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 gao 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)

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
```

Loại giao dịch

- Có 3 loại giao dịch:
  - **Explicit** transactions: giao dịch tường minh
  - **Autocommit** transactions: giao dịch tự động
  - **Implicit** transactions: giao dịch ngầm

Explicit transactions

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

Explicit transactions

- **Tạo điểm lưu** (SAVE POINT)

  save tran tên_điểm_lưu

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

Điểm lưu (save point)

- **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**

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

Autocommit Transactions

- **Au*tocommit* mode**: mode 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.

Implicit transactions

- Giao dịch ngầm định, được thiết lập khi có một kết nối đến CSDL, tạo ra một chuỗi các giao dịch, tự động commit hoặc rollback từng giao dịch.
- **Bắt đầu giao tác với các lệnh** ALTER TABLE, DROP, TRUNCATE TABLE, CREATE, OPEN, FETCH, REVOKE, GRANT DELETE, INSERT, SELECT, UPDATE
- **Kết thúc bằng lệnh : commit | rollback tran**

Implicit transactions

- Thiết lập thông số :

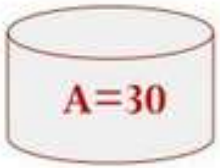
  SET IMPLICIT_TRANSACTIONS ON|OFF

Các vấn đề truy xuất 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.

Mất dữ liệu khi cập nhật - Lost updated

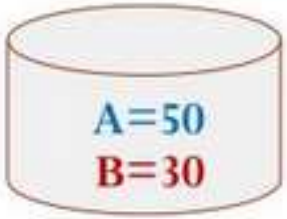
- Xét ví dụ:
  - P_1 và P_2 xử lý đồng thời:



	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	

Đọc dữ liệu rác - Dirtty Read

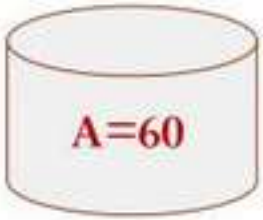
- P_1 và P_2 xử lý đồng thời:



	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	

Không thể đọc lại - Unrepeatable Read

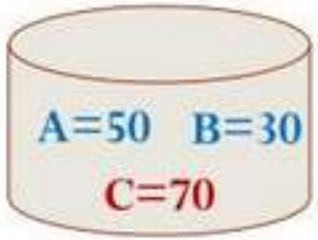
- P_1 và P_2 xử lý đồng thời:



	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		

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	

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)

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 thanh 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

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

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 xãy 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:

KHACHHANG

MaKH	TenKH	DiaChi	DienThoai
KH001	ABC Văn Tuyên	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

Slock

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

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 giữa 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

```
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
```

T2

```
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 thiêt lập Xlock trên đơn vị dữ liệu được ghi và Xlock được giữ cho đến khi giao dịch kết thức

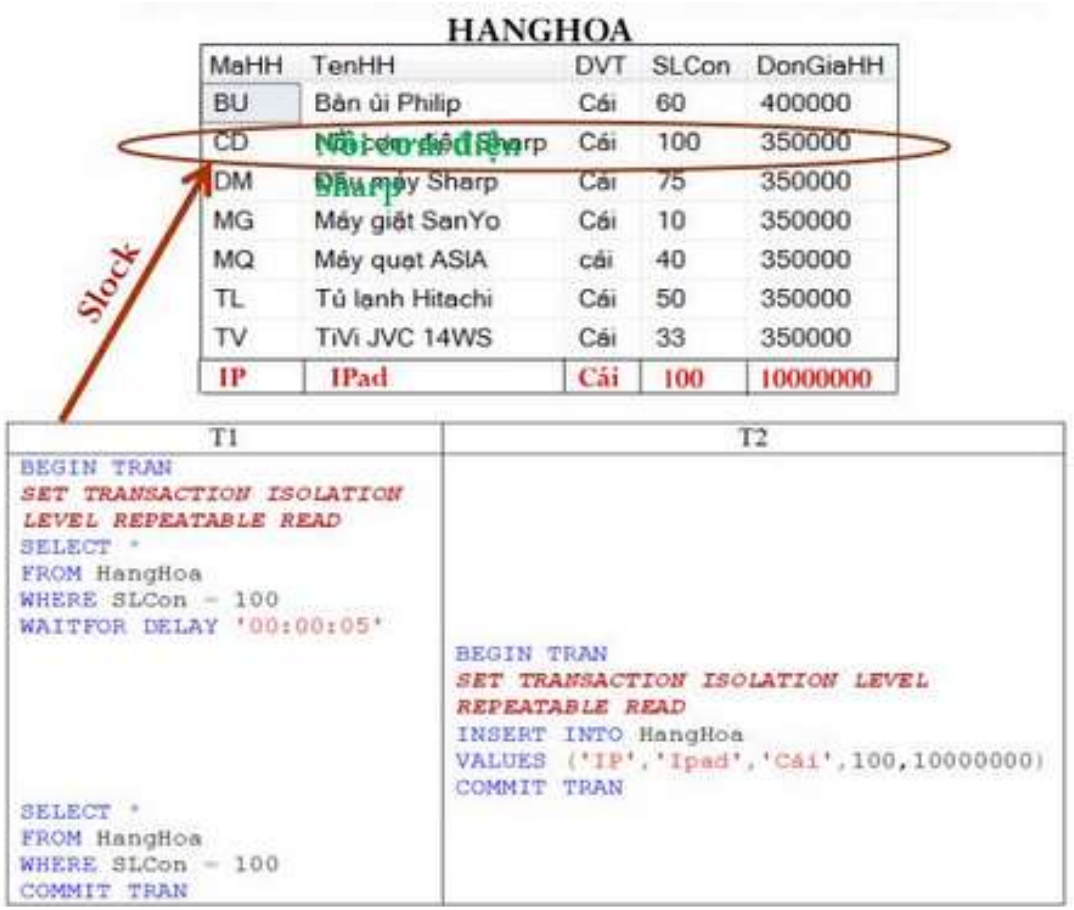
Serializable

- 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

Đặt vấn đề

- Các mức cô lập quyết định các phát và giữa khóa Slock trong một giao tác và có hiệu lực trên tất cả các thao tác đọc trong giao tác đó
- Thực tế, cần phát và giữa khóa Slock theo các cách khác nhau cho các thao rác đọc khác nhau torng cùng một giao dịch

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 ...
```