



# Quản Trị Dữ Liệu Với Microsoft SQL Server

## Chương: 14

### Giao dịch Transactions

- Định nghĩa và mô tả giao dịch (transaction)
- Giải thích các thủ tục để thực hiện giao dịch
- Giải thích quá trình kiểm soát các giao dịch
- Giải thích các bước để đánh dấu một giao dịch
- Phân biệt giữa giao dịch ngầm định(implicit) và tường minh(explicit)
- Giải thích các mức cô lập
- Giải thích phạm vi và các loại khóa khác nhau
- Giải thích quản lý giao dịch

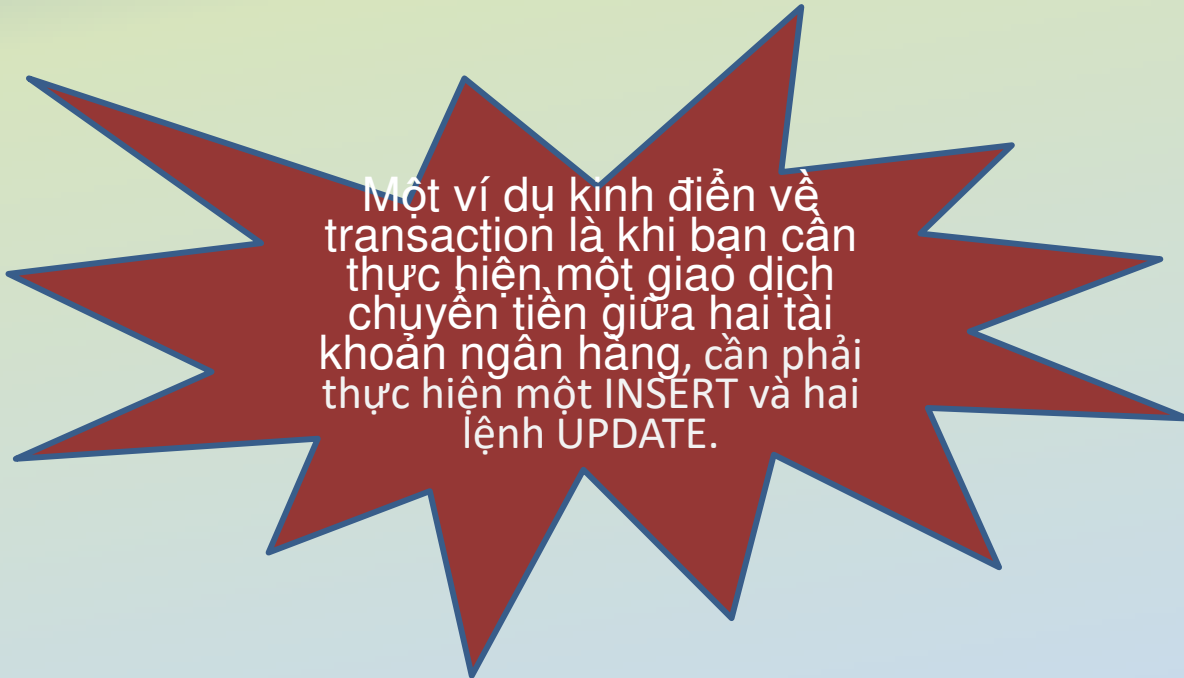
- Một transaction(giao dịch) là
  - một công việc đơn (unit of work).
  - là thành công chỉ khi tất cả sự thay đổi dữ liệu được thực hiện trong một transaction được hoàn tất(commit) và được lưu trong cơ sở dữ liệu vĩnh viễn.
- Nếu giao dịch bị quay lại trạng thái ban đầu(rolled) hoặc hủy bỏ, thì có nghĩa là transaction(giao dịch) đã gặp lỗi và không xảy ra sự thay đổi nào với các nội dung của cơ sở dữ liệu.
- Một transaction có thể được hoàn tất (commit) hoặc quay về trạng thái ban đầu (roll back).

# Sự cần thiết của Transaction 1-5

Có rất nhiều trường hợp người sử dụng cần phải thực hiện nhiều thay đổi dữ liệu trong nhiều bảng của cơ sở dữ liệu.

Trong vài trường hợp, dữ liệu sẽ không nhất quán khi thực thi các lệnh riêng biệt.

Giả sử nếu câu lệnh đầu tiên thực hiện một cách chính xác nhưng các câu lệnh khác thì thất bại, lúc này dữ liệu sẽ trong trạng thái sai lệch(incorrect).



Một ví dụ kinh điển về transaction là khi bạn cần thực hiện một giao dịch chuyển tiền giữa hai tài khoản ngân hàng, cần phải thực hiện một INSERT và hai lệnh UPDATE.

## Sự cần thiết của Transaction 2-5

- Một ví dụ kinh điển về transaction là khi bạn cần thực hiện một giao dịch chuyển tiền giữa hai tài khoản ngân hàng. Giả sử bạn có hai tài khoản A và B với số tiền tương ứng là 8 tỷ và 1 tỷ; nay bạn cần chuyển bớt 2 tỷ từ tài khoản A sang tài khoản B. Sẽ có hai phép cập nhật như sau:
  - trừ số tiền hiện có của tài khoản A đi 2 tỷ
  - cộng thêm số tiền hiện có của tài khoản B lên 2 tỷ
  
- Nếu hai lệnh cập nhật trên diễn ra độc lập (không nằm trong một transaction), và vì một lý do nào đó lệnh thứ hai bị lỗi, tài khoản A sẽ còn 6 tỷ và tài khoản B vẫn giữ nguyên 1 tỷ. Điều này không thể chấp nhận được vì 2 tỷ bỗng dưng biến mất! Khi thực hiện hai lệnh trên trong một transaction, nó sẽ đảm bảo:
  - hoặc cả hai lệnh update đều được thực hiện thành công. Cả hai tài khoản được cập nhật với số tiền tương ứng.
  - hoặc trong trường hợp giao dịch bị lỗi cả hai lệnh đều không được thực hiện. Hai tài khoản giữ nguyên số tiền như trước khi thực hiện transaction.

# Sự cần thiết của Transaction 3-5

**Định nghĩa Transactions:** Một khối công việc(unit of work) phải thể hiện (exhibit) bốn thuộc tính ACID là: nguyên tử (atomicity), nhất quán (consistency), cô lập (isolation), và lâu bền (durability) , thì đủ điều kiện là một giao dịch.

**Nguyên tử (atomicity) :**  
Nếu transaction có nhiều thao tác thì tất cả chúng nên được hòa tất.

**Nhất quán (Consistency):** Dãy các thao tác phải nhất quán.

**Cô lập (Isolation):** Các thao tác phải được thực hiện cô lập khỏi các thao tác khác trên cùng server hoặc trên cùng csdl.

**Lâu bền (Durability):** Các thao tác được thực hiện trên csdl phải được save và lưu trữ trong csdl vĩnh viễn.

# Sự cần thiết của Transaction 4-5

**Thực thi transactions:** SQL Server hỗ trợ các giao dịch ở nhiều chế độ.

- **Giao dịch hoàn tất tự động(Autocommit Transactions):** Mỗi câu lệnh đơn được hoàn tất một cách tự động ngay khi nó được hoàn thành.
- **Giao dịch tường minh(Explicit Transactions):** Mọi transaction bắt đầu tường minh với câu lệnh `BEGIN TRANSACTION` và kết thúc bằng câu lệnh `ROLLBACK` hoặc `COMMIT`.
- **Giao dịch ngầm định(Implicit Transactions):** Một transaction mới được bắt đầu tự động khi transaction trước đó hoàn thành, và mỗi giao dịch được hoàn tất tường minh bằng việc sử dụng lệnh `ROLLBACK` hoặc `COMMIT`.
- **Các giao dịch phạm vi trong batch(Batch-scoped Transactions):** Đây là các giao dịch có liên quan đến Multiple Active Result Sets (MARS).



# Sự cần thiết của Transaction 5-5

## ➤ Transactions Extending Batches

Các lệnh transaction xác định khối code hoặc thất bại hoặc thành công và cung cấp cơ sở cho bộ máy cơ sở dữ liệu có thể quay lùi lại ban đầu(rollback) hoặc quay lại(undo) các thao tác.

Người dùng có thể thêm code để xác định khối(batch) như một giao dịch và đặt khối(batch) giữa `BEGIN TRANSACTION` và `COMMIT TRANSACTION`.

Người dùng có thể thêm đoạn code xử lý lỗi để quay lui lại trạng thái ban đầu trong trường hợp giao dịch lỗi. Đoạn code xử lý lỗi sẽ quay lại(undo) các thay đổi đã được thực hiện về trạng thái trước khi có lỗi xảy ra



# Kiểm soát transaction

Các giao dịch có thể được kiểm soát thông qua các ứng dụng bằng cách xác định điểm bắt đầu và kết thúc của một giao dịch.

Điều này được thực hiện bằng cách sử dụng các hàm API cơ sở dữ liệu hoặc các câu lệnh Transact-SQL.

Khi một giao dịch được bắt đầu trên một kết nối, tất cả các câu lệnh Transact-SQL được thực hiện trên cùng một kết nối và là một phần của kết nối cho đến khi giao dịch kết thúc.

Giao dịch được quản lý ở cấp kết nối.



# Bắt đầu và kết thúc phiên giao dịch bằng Transact-SQL 1-11

Một trong nhiều cách người dùng có thể bắt đầu và kết thúc các transactions là bằng các câu lệnh Transact-SQL.

Người dùng có thể bắt đầu một transaction trong SQL Server theo các chế độ ngầm định (implicit) hoặc tường minh (explicit).

Chế độ giao dịch tường minh (Explicit transaction mode) bắt đầu một giao dịch bằng câu lệnh `BEGIN TRANSACTION`.

Người dùng có thể kết thúc giao dịch bằng câu lệnh `ROLLBACK` hoặc `COMMIT`.

# Bắt đầu và kết thúc phiên giao dịch bằng Transact-SQL 2-11

Câu lệnh `BEGIN TRANSACTION` đánh dấu điểm bắt đầu của một transaction tường minh hoặc cục bộ.

## Cú pháp:

```
BEGIN { TRAN | TRANSACTION }  
[ { transaction_name | @tran_name_variable }  
[ WITH MARK [ 'description' ] ]  
] [ ; ]
```

Trong đó,

`transaction_name`: chỉ ra tên được gán cho transaction specifies the name that is assigned to the transaction. Nó nên tuân theo quy tắc định danh và giới hạn độ dài là 32 kí tự.

`@tran_name_variable`: chỉ ra tên biến do người dùng định nghĩa có chứa tên transaction hợp lệ.

`WITH MARK [ 'description' ]`: chỉ ra transaction được đánh dấu trong log. Chuỗi mô tả (description string) định nghĩa điểm đánh dấu.



# Bắt đầu và kết thúc phiên giao dịch bằng Transact-SQL

## 3-11

- Đoạn code dưới đây cho thấy cách tạo và bắt đầu một transaction:

```
USE AdventureWorks2012;  
GO  
DECLARE @TranName VARCHAR(30);  
SELECT @TranName = 'FirstTransaction';  
BEGIN TRANSACTION @TranName;  
    DELETE FROM HumanResources.JobCandidate  
    WHERE JobCandidateID = 13;
```



# Bắt đầu và kết thúc phiên giao dịch bằng Transact-SQL 4-11

Câu lệnh `COMMIT TRANSACTION` đánh dấu kết thúc một transaction ngầm định hoặc tường minh thành công.

## Cú pháp:

```
COMMIT { TRAN | TRANSACTION } [ transaction_name |  
    @tran_name_variable ] [ ; ]
```

Trong đó,

`transaction_name`: chỉ ra tên transaction đã được gán ở câu lệnh `BEGIN TRANSACTION` trước đó.

`@tran_name_variable`: chỉ ra tên biến do người dùng định nghĩa có chứa tên transaction hợp lệ. Biến có thể khai báo kiểu dữ liệu `char`, `varchar`, `nchar`, hoặc `nvarchar`. Nếu gán cho biến một chuỗi nhiều hơn 32 kí tự, thì chỉ có 32 kí tự được sử dụng còn lại sẽ bị cắt bỏ.

# Bắt đầu và kết thúc phiên giao dịch bằng Transact-SQL 5-11

- Đoạn code dưới đây cho thấy cách hoàn tất (commit) một transaction:

```
BEGIN TRANSACTION;  
GO  
DELETE FROM HumanResources.JobCandidate  
WHERE JobCandidateID = 11;  
GO  
COMMIT TRANSACTION;  
GO
```

Câu lệnh COMMIT WORK đánh dấu kết thúc một transaction.

## Cú pháp:

```
COMMIT [ WORK ] [ ; ]
```

COMMIT TRANSACTION và COMMIT WORK là giống hệt nhau, ngoại trừ một thực tế là COMMIT TRANSACTION chấp nhận một tên giao dịch người dùng định nghĩa.

# Bắt đầu và kết thúc phiên giao dịch bằng Transact-SQL 6-11

## Đánh dấu một Transaction

- Đoạn code sau đây cho thấy cách đánh dấu một transaction:

```
BEGIN TRANSACTION DeleteCandidate  
WITH MARK N'Deleting a Job Candidate';  
GO  
DELETE FROM HumanResources.JobCandidate  
WHERE JobCandidateID = 11;  
GO  
COMMIT TRANSACTION DeleteCandidate;
```

ROLLBACK TRANSACTION – Đây là hủy bỏ(cancel) hoặc quay lui (roll back) một transaction ngầm định (implicit) hoặc tường minh (explicit) về trạng thái lúc bắt đầu hoặc về điểm savepoint.

## Cú pháp:

```
COMMIT [ WORK ] [ ; ]
```



# Bắt đầu và kết thúc phiên giao dịch bằng Transact-SQL

## 7-11

**ROLLBACK TRANSACTION** - Giao dịch ngầm định hoặc giao dịch tường minh này bị hủy bỏ hay quay trở lại thời điểm ban đầu của giao dịch, hoặc tới điểm savepoint trong một giao dịch.

```
ROLLBACK { TRAN | TRANSACTION }
[ transaction_name | @tran_name_variable
| savepoint_name | @savepoint_variable ] [ ; ]
```

Trong đó,

**transaction\_name:** chỉ ra tên được gán cho lệnh **BEGIN TRANSACTION**. Tên được tuân thủ theo qui tắc định danh.

**@tran\_name\_variable:** chỉ ra tên của biến người dùng định nghĩa, biến này lưu trữ tên một tên giao dịch hợp lệ. Biến có thể được khai báo các kiểu dữ liệu **char**, **varchar**, **nchar**, hoặc **nvarchar**.

**savepoint\_name:** chỉ ra tên điểm **savepoint\_name** từ câu lệnh **SAVE TRANSACTION**. Chỉ dùng **savepoint\_name** khi điều kiện rollback tác động đến một phần giao dịch.

**@savepoint\_variable:** chỉ ra của tên biến **savepoint**, là biến có chứa tên **savepoint** hợp lệ. Biến có thể được khai báo với kiểu dữ liệu **char**, **varchar**, **nchar**, hoặc **nvarchar**.



# Bắt đầu và kết thúc phiên giao dịch bằng Transact-SQL 8-11

- Đoạn code sau minh họa sử dụng ROLLBACK:

```
USE Sterling;  
GO  
CREATE TABLE ValueTable ([value] char)  
GO
```

- Đoạn code sau đây cho thấy một giao tác chèn hai bản ghi vào bảng ValueTable:

```
BEGIN TRANSACTION  
INSERT INTO ValueTable VALUES('A');  
INSERT INTO ValueTable VALUES('B');  
GO  
ROLLBACK TRANSACTION  
INSERT INTO ValueTable VALUES('C');  
SELECT [value] FROM ValueTable;
```

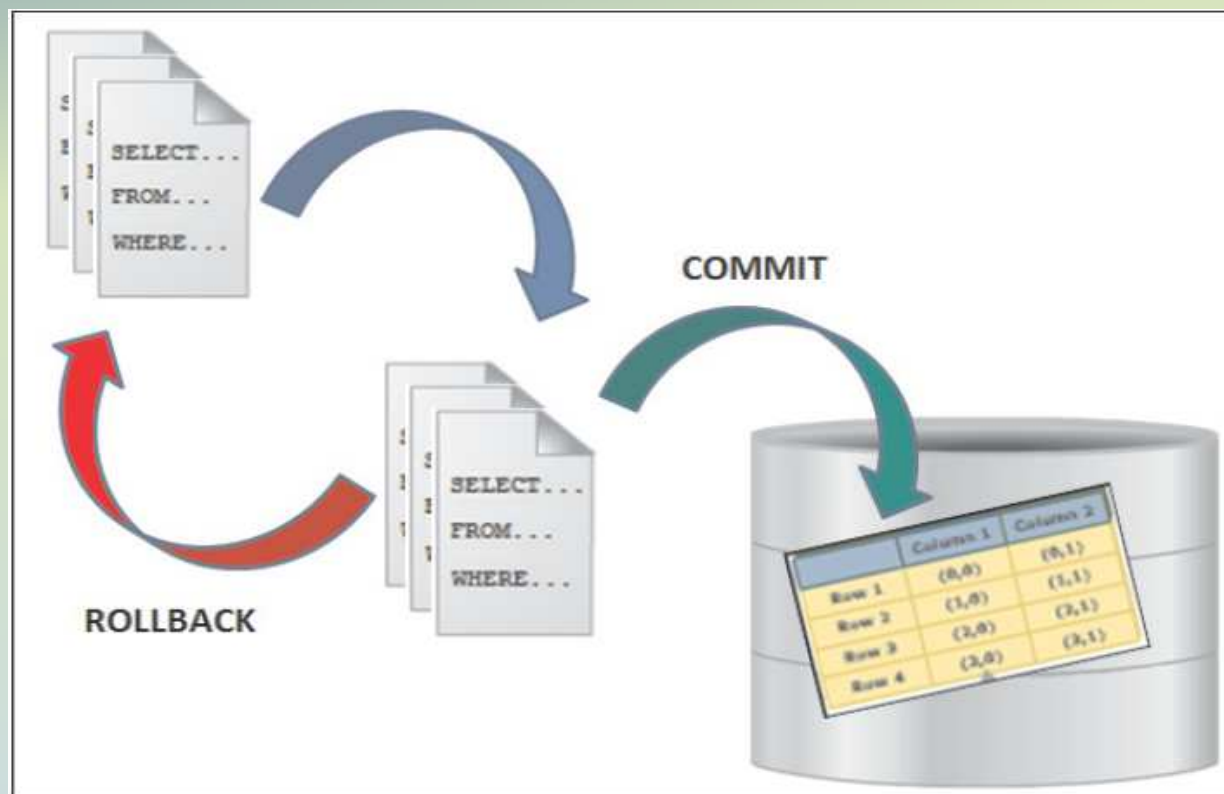
- Sau đó, nó rolls back giao dịch và lại chèn một bản ghi vào bảng ValueTable.

# Bắt đầu và kết thúc phiên giao dịch bằng Transact-SQL 9-11

Lệnh **ROLLBACK WORK** thực hiện quay lui giao dịch do người dùng chỉ ra về thời điểm bắt đầu của giao dịch.

## Cú pháp:

```
ROLLBACK [ WORK ] [ ; ]
```





# Bắt đầu và kết thúc phiên giao dịch bằng Transact-SQL 10-11

Lệnh `SAVE TRANSACTION` thiết lập một điểm savepoint bên trong một giao dịch.

## Cú pháp:

```
SAVE { TRAN | TRANSACTION } { savepoint_name | @savepoint_variable }  
[ ; ]
```

trong đó,

`savepoint_name`: chỉ ra tên được gán với điểm `savepoint_name`. Các tên được đặt theo quy tắc định danh.

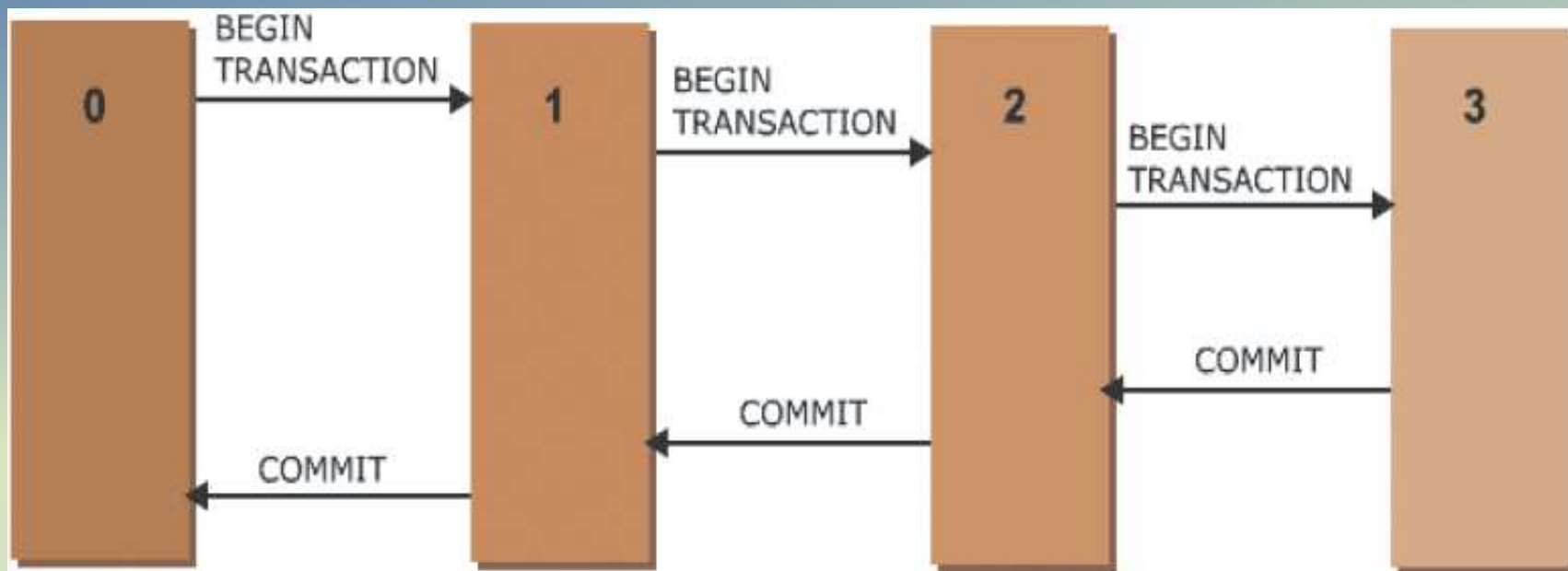
`@savepoint_variable`: chỉ ra tên của biến do người dùng định nghĩa có chứa tên điểm savepoint hợp lệ. Biến có thể được khai báo với kiểu dữ liệu `char`, `varchar`, `nchar`, hoặc `nvarchar`. Cho phép gán một chuỗi nhiều hơn 32 kí tự cho biến, nhưng chỉ có 32 kí tự đầu tiên được sử dụng.

- Đoạn code sau đây minh họa cách dùng savepoint transaction:

```
CREATE PROCEDURE SaveTranExample
    @InputCandidateID INT
AS
    DECLARE @TranCounter INT;
    SET @TranCounter = @@TRANCOUNT;
    IF @TranCounter > 0
        SAVE TRANSACTION ProcedureSave;
    ELSE
        BEGIN TRANSACTION;
        DELETE HumanResources.JobCandidate
            WHERE JobCandidateID = @InputCandidateID;
        IF @TranCounter = 0
            COMMIT TRANSACTION;
        IF @TranCounter = 1
            ROLLBACK TRANSACTION ProcedureSave;
GO
```

# @@TRANCOUNT 1-2

Hàm hệ thống @@TRANCOUNT trả về một tổng số các câu lệnh BEGIN TRANSACTION xuất hiện(occur) trong kết nối hiện tại.



**Cú pháp:**

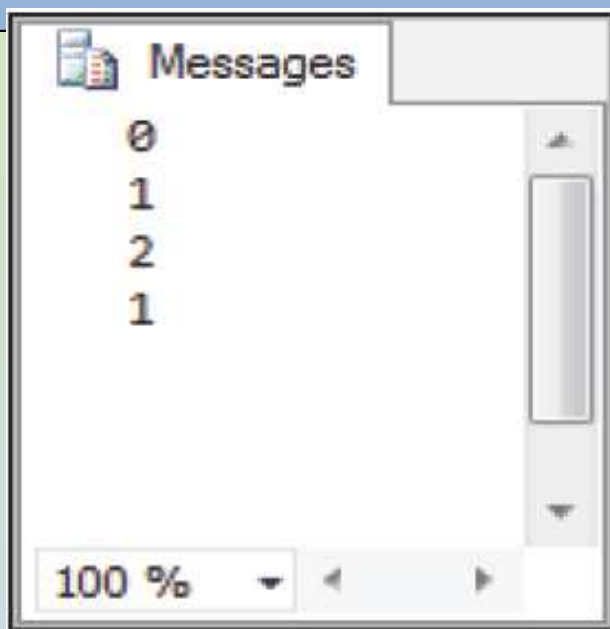
```
@@TRANCOUNT
```

# @@TRANCOUNT 2-2

- Đoạn code sau đây cho thấy tác dụng(effect) mà các câu lệnh BEGIN và COMMIT lồng nhau có biến @@TRANCOUNT:

```
PRINT @@TRANCOUNT  
BEGIN TRAN  
    PRINT @@TRANCOUNT  
    BEGIN TRAN  
        PRINT @@TRANCOUNT  
    COMMIT  
    PRINT @@TRANCOUNT  
COMMIT  
PRINT @@TRANCOUNT
```

**Kết quả:**





# Đánh dấu Transaction 1-3

Người dùng có thể sử dụng các nhãn giao dịch(transaction marks) để phục hồi các bản cập nhật có liên quan thực hiện các cập nhật(update) tới hai hoặc nhiều cơ sở dữ liệu.

Đánh dấu một giao dịch chỉ có ích khi người dùng muốn làm mất các giao dịch đã được hoàn tất gần đây hoặc đang thử nghiệm các cơ sở dữ liệu quan hệ.

Đánh dấu các giao dịch có liên quan trên một chương trình con(trigger, stored procedure) cơ bản trong mỗi csdl quan hệ duy nhất tạo ra một chuỗi các điểm phục hồi thông thường trong một cơ sở dữ liệu.

Các nhãn giao dịch(transaction marks) được kết hợp trong sao lưu log và cũng được ghi lại trong nhật ký giao dịch (transaction log).

Trong trường hợp có thảm họa, người dùng có thể khôi phục mỗi cơ sở dữ liệu tới cùng vị trí đánh dấu giao dịch để phục hồi chúng đến một điểm phù hợp.

## Đánh dấu Transaction 2-3

**Cần nhắc việc sử dụng các transaction được đánh dấu.**

Trước khi chèn tên đánh dấu(named marks) vào nhật ký giao dịch, hãy xem xét sau đây:

- Do nhãn đánh dấu giao dịch tiêu tốn không gian log, nên chỉ sử dụng chúng cho các transaction đóng một vai trò quan trọng trong chiến lược phục hồi cơ sở dữ liệu.
- Khi transaction được đánh dấu hoàn tất, sau đó một dòng được chèn vào bảng `logmarkhistory` trong `msdb`.
- Nếu một giao dịch được đánh dấu dàn trải qua nhiều cơ sở dữ liệu trên các máy chủ khác nhau hoặc trên cùng máy chủ cơ sở dữ liệu, các nhãn đánh dấu(mark) phải được log trong các bản ghi của tất cả các cơ sở dữ liệu bị ảnh hưởng.

## Đánh dấu Transaction 3-3

- Để tạo một transaction có nhãn đánh dấu(marked transaction) trong một tập csdl, cần tuân thủ theo các bước như sau:
  1. Tên transaction trong câu lệnh BEGIN TRANSACTION và sử dụng mệnh đề WITH MARK
  2. Chạy update lại tất cả các csdl có trong tập hợp
- Đoạn code dưới đây minh họa việc tạo một marked transaction trong một tập csdl:

```
USE AdventureWorks2012
GO
BEGIN TRANSACTION ListPriceUpdate
    WITH MARK 'UPDATE Product list prices';
GO
UPDATE Production.Product
    SET ListPrice = ListPrice * 1.20
    WHERE ProductNumber LIKE 'BK-%';
GO
COMMIT TRANSACTION ListPriceUpdate;
GO
```



# Sự khác biệt giữa transaction ngầm định và tường minh

- Bảng sau đây liệt kê sự khác biệt giữa transaction ngầm định và tường minh:

Implicit	Explicit
These transactions are maintained by SQL Server for each and every DML and DDL statements	These transactions are defined by programmers
These DML and DDL statements execute under the implicit transactions	DML statements are included to execute as a unit
SQL Server will roll back the entire statement	SELECT Statements are not included as they do not modify data

# Các mức cô lập (Isolation Levels) 1-2

Các transaction xác định các mức độ cô lập (định nghĩa cấp độ) mà một giao dịch phải được cách ly(isolated) khỏi các thao tác sửa đổi dữ liệu hoặc tài nguyên được thực hiện bởi các giao dịch khác.

Mức độ cô lập được định nghĩa có liên quan tới khía cạnh xử lý đồng thời (concurrent) như là dirty reads(đọc dữ liệu chưa commit) được cho phép.

Khi một giao dịch thay đổi một giá trị và một giao dịch thứ hai cùng đọc giá trị đó trước khi thay đổi ban đầu đã được hoàn tất hoặc rollback, nó được gọi là một dirty read.

Một giao dịch chiếm được khóa độc quyền(exclusive lock) trong toàn bộ thời gian trên mỗi dữ liệu mà nó sửa đổi và nó khóa cho đến khi giao dịch được hoàn thành, không phân biệt mức độ cô lập được thiết lập cho giao dịch đó.

Một mức độ cô lập thấp hơn làm tăng khả năng của nhiều người dùng truy cập dữ liệu tại cùng thời điểm. Một mức độ cô lập cao hơn làm giảm hiệu quả truy cập đồng thời mà người dùng có thể gặp phải.

## Các mức cô lập (Isolation Levels) 2-2

- Bảng dưới đây liệt kê các ảnh hưởng của xử lý đồng thời (concurrency effects) của từng mức isolation.

Isolation Level	Dirty Read	NonRepeatable Read
Read committed	No	Yes
Read uncommitted	Yes	No
Snapshot	No	No
Repeatable Read	No	No
Serializable	No	No

# Phạm vi và các loại khóa khác nhau 1-6

SQL Server Database Engine khóa các tài nguyên bằng việc sử dụng chế độ khóa khác nhau, trong đó xác định xem các tài nguyên nào có thể truy cập đến các giao dịch đồng thời.

➤ Bảng dưới đây liệt kê các chế độ khóa được sử dụng bởi Database Engine:

Lock Mode	Description
Update	Is used on resources that are to be updated.
Shared	Is used for read operations that do not change data such as SELECT statement.
Intent	Is used to establish a hierarchy of locks.
Exclusive	Is used for INSERT, UPDATE, or DELETE data-modification operations.
BULK UPDATE	Is used while copying bulk data into the table.
Schema	Is used when the operation is dependent on the table schema.



# Phạm vi và các loại khóa khác nhau 2-6

## Update Locks

- tránh hình thức của deadlock(bế tắc) phổ biến.
- Khi hai giao dịch có được khóa chia sẻ trên một nguồn tài nguyên và cố gắng cập nhật dữ liệu cùng một lúc, cùng một giao dịch cố gắng chuyển đổi khóa thành khóa độc quyền(exclusive lock)
- Chỉ có một giao dịch có thể lấy được khóa cập nhật cho một tài nguyên tại một thời điểm
- Khi giao dịch thực hiện thay đổi nguồn tài nguyên, thì các khóa update (update lock) các khóa cập nhật được chuyển thành khóa độc quyền (exclusive lock)

## Shared Locks

- cho phép các giao dịch chạy song song để cùng đọc một tài nguyên dưới sự kiểm soát đồng thời bi quan (pessimistic concurrency control)
- Giao dịch có thể thay đổi dữ liệu trong khi khóa chia sẻ(shared locks) tồn tại trên tài nguyên.
- Khóa chia sẻ được giải phóng trên tài nguyên mỗi khi thao tác đọc hoàn thành

# Phạm vi và các loại khóa khác nhau 3-6

## Exclusive Locks

ngăn chặn các cùng giao dịch cùng đồng thời truy cập đến tài nguyên.

không có giao dịch khác nào khác có thể thay đổi dữ liệu và chỉ có thao tác đọc diễn ra thông qua mức độ cô lập không read uncommitted hoặc gợi ý NOLOCK

Các câu lệnh DML thường yêu cầu cả hai là khóa khóa độc quyền(exclusive) và khóa chia sẻ(shared lock)

Khi giao dịch thực hiện thay đổi một nguồn tài nguyên, thì các khóa update lock được chuyển thành khóa exclusive lock

## Intent Locks

- được Database Engine sử dụng để bảo vệ và việc đặt một khóa độc quyền (exclusive) hoặc khóa chia sẻ(shared lock) trên tài nguyên mức thấp hơn trong hệ thống phân cấp khóa
- giành được trước khi một khóa ở mức thấp và do đó, chỉ ra mục đích đặt khóa ở mức thấp
- sử dụng cho hai mục đích:
  - ngăn chặn các giao dịch khác khỏi việc thay đổi nguồn mức cao hơn
  - nâng cao hiệu quả của Database Engine cho việc xác định các xung đột khóa.

# Phạm vi và các loại khóa khác nhau 4-6

Thiết lập các khóa dự định(intent lock) ở cấp độ bảng bảo vệ giao dịch khác khỏi về sau giành được một khóa độc quyền(exclusive) trên các bảng đang chứa các trang(page).

➤ Bảng dưới đây liệt kê các chế độ khóa trong khóa dự định (Intent locks):

Lock Mode	Description
Intent shared (IS)	Protects the requested shared lock on some resources that are lower in the hierarchy.
Intent exclusive (IX)	Protects the requested exclusive lock on some resources lower in the hierarchy. IX is a superset of IS, that protects requesting shared locks on lower level resources.
Shared with Intent Exclusive (SIX)	Protects the requested shared lock on all resources lower in the hierarchy and intent exclusive locks on some of the lower level resources. Concurrent IS locks are allowed at the top-level resource.
Intent Update (IU)	Protects the requested update locks on all resources lower in the hierarchy. IU locks are used only on page resources. IU locks are converted to IX locks if an update operation takes place.
Shared intent update (SIU)	Provides the combination of S and IU locks, as a result of acquiring these locks separately and simultaneously holding both locks.
Update intent exclusive (UIX)	Provides the combination of U and IX locks, as a result of acquiring these locks separately and simultaneously holding both locks.

# Phạm vi và các loại khóa khác nhau 5-6

- Có nhiều người liên quan đến thiết kế, sử dụng và bảo trì(maintenance ) một csdl lớn với hàng trăm người dùng.

## Bulk Update Locks

- Database Engine sử dụng khóa bulk update (BU) khi một khối dữ liệu lớn được copy vào một bảng (thao tác copy hàng loạt bằng dòng lệnh bcp ) và hoặc `TABLOCK` hoặc tùy chọn bảng **table lock on bulk load** được thiết lập bằng việc dùng `sp_tableoption`
- Cho phép nhiều luồng (multiple threads) tải dữ liệu hàng loạt (load bulk data) đồng thời vào cùng bảng, trong khi ngăn cản các tiến trình khác(processes) tải hàng loạt dữ liệu từ bảng đang truy cập

## Schema Locks

- được sử dụng bởi Database Engine trong khi thực hiện các lệnh DDL như xóa một bảng hoặc một cột
- ngăn cản truy cập đồng thời tới bảng, có nghĩa là khóa lược đồ(schema lock) ngăn chặn(block) tất cả các thao tác bên ngoài cho tới khi khóa được giải phóng(releases)
- được sử dụng bởi Database Engine trong khi biên dịch và thực thi các truy vấn.

# Phạm vi và các loại khóa khác nhau 6-6

## Key-Range Locks

- bảo vệ một tập (collection) các dòng có trong một recordset(tập các bản ghi) đang đọc bởi câu lệnh Transact-SQL trong khi sử dụng mức cô lập giao dịch serializable
- ngăn cản việc chèn hoặc xóa phantom trong recordset(tập các bản ghi) mà giao dịch truy cập

# Quản lý Transaction

Mặc định mỗi câu lệnh đơn được thực thi như là một, transactional trong SQL Server.

Nếu một câu lệnh SQL đơn được đưa ra, thì một giao dịch ngầm định được bắt đầu.

Khi người dùng sử dụng các câu lệnh `BEGIN TRAN/COMMIT TRAN` tường minh, chúng có thể nhóm chúng lại với nhau như là một giao dịch tường minh.

SQL Server thực thi nhiều mức cô lập giao dịch đảm bảo thuộc tính ACID của các giao dịch này.

Tuy nhiên, dữ liệu được lưu trữ trong dạng này là không lâu bền. Các bản ghi lưu trong các tập tin thủ công chỉ có thể duy trì trong vài tháng hoặc vài năm.



# Lưu vết giao dịch (Transaction Log) 1-3

Cơ sở dữ liệu SQL Server có lưu vết giao dịch, ghi lại tất cả các giao dịch và việc chỉnh sửa csdl được thực hiện bởi mỗi giao dịch.

Nhật ký giao dịch(transaction log ) nên được cắt xén(truncate) thường xuyên để giữ cho nó khỏi bị đầy (filling up).

Các thao tác được hỗ trợ bởi lưu vết giao dịch như sau:

- Phục hồi riêng biệt các giao dịch
- Phục hồi các giao không hoàn thành(Incomplete) khi SQL Server được khởi động
- Hỗ trợ nhân bản(replication) giao dịch.
- Các giải pháp phục hồi thảm họa và hỗ trợ tính sẵn sàng cao
- Roll back một tập tin, khôi phục (restored) cơ sở dữ liệu, filegroup, hoặc page tiến tới điểm lỗi.



# Lưu vết giao dịch (Transaction Log) 2-3

## ➤ Việc cắt(truncate) một nhật ký giao dịch (transaction log)

Cắt bớt nhật ký(log) để giải phóng không gian trong các tập tin log để tái sử dụng nhật ký giao dịch.

Việc cắt bớt nhật ký tự động khởi động sau các sự kiện dưới đây:

- Trong một mô hình phục hồi đơn giản sau khi checkpoint
- Trong một mô hình phục hồi bulk-logged hoặc mô hình phục hồi full, nếu các checkpoint được xảy ra kể từ khi sao lưu cuối cùng, cắt bớt xảy ra sau khi một sao lưu nhật ký

# Lưu vết giao dịch (Transaction Log) 3-3

Cắt giảm nhật ký bị trì hoãn vì nhiều lý do. Người dùng cũng có thể khám phá nếu bất cứ điều gì làm ngăn cản sự cắt giảm nhật ký bằng cách truy vấn các cột `log_reuse_wait_desc` và `log_reuse_wait` của view `sys.databases`.

➤ Bảng sau đây liệt kê giá trị của các cột sau:

Log_reuse_wait mode	Log_reuse_wait value	desc	Description
0	NOTHING		Specifies that at present, there are more than one reusable virtual log file.
1	CHECKPOINT		Specifies that there is no checkpoint occurred since the last log truncation, or the head of the log has not moved beyond a virtual log file.
2	LOG_BACKUP		Specifies a log backup that is required before the transaction log truncates.
3	ACTIVE_BACKUP_OR_RESTORE		Specifies that the data backup or a restore is in progress.
4	ACTIVE_TRANSACTION		Specifies that a transaction is active.
5	DATABASE_MIRRORING		Specifies that the database mirroring is paused, or under high-performance mode, the mirror database is significantly behind the principal database.

- Một giao dịch là một dãy các thao tác làm việc như là một khối duy nhất.
- Các giao dịch có thể được điều khiển bởi một ứng dụng bằng việc chỉ ra bắt đầu và kết thúc.
- BEGIN TRANSACTION đánh dấu điểm bắt đầu của một giao dịch tường minh hoặc cục bộ.
- COMMIT TRANSACTION đánh dấu kết thúc thành công giao dịch tường minh hoặc giao dịch ngầm định.
- ROLLBACK với tùy khóa tùy chọn WORK là để quay lui lại (rolls back) một giao dịch người dùng định nghĩa về thời bắt đầu của giao dịch.
- @@TRANCOUNT là hàm hệ thống trả về tổng số câu lệnh BEGIN TRANSACTION xuất hiện trong kết nối hiện hành.
- Mức cô lập(Isolation levels) được cung cấp bởi giao dịch để mô tả phạm vi(extent) mà một giao dịch đơn cần được cách ly(isolated) khỏi sự thay đổi được thực hiện bởi các giao dịch khác.
- SQL Server Database Engine khóa các tài nguyên bằng việc sử dụng các chế độ khóa khác nhau, để xác định xem tài nguyên có thể truy cập tới các giao dịch đang chạy đồng thời (concurrent transactions).