HƯỚNG DẪN THỰC HÀNH
HỆ QUẢN TRỊ CSDL

1. SỬ DỤNG TRANSACTION

1.1. Khái niệm transaction :

Giao tác (transaction) là 1 tập hợp có thứ tự các thao tác (statement) truy xuất dữ liệu trên CSDL thành 1 đơn vị công việc logic (xem là 1 thao tác *nguyên tố),* chuyển CSDL từ trạng thái nhất quán này sang trạng thái nhất quán khác.

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 A, tăng tiền của B. Hai công việc này hoặc cả hai thành công hoặc không có công việc nào thành công (nếu một công việc vì lý do nào đó không thực hiện thành công thì trạng thái ban đầu trước khi chuyển tiền phải được khôi phục để bảo toàn dữ liệu). Khi đó việc chuyển tiền cần được đặt vào một giao tác.

Chú ý : khi ta viết một thao tác (statement) trong SQL Server, nếu không có chỉ thị nào khác, thao tác này là một transaction.

1.2. Tại sao phải dùng transaction ?

- Dùng khái niệm giao tác khi xử lý các vấn đề liên quan đến truy xuất dữ liệu đồng thời
- Có những xử lý trên CSDL được thực hiện bằng nhiều thao tác liên tiếp nhau, tập hợp các thao tác này phải được xem là *một thao tác nguyên tố* để đảm bảo tính nhất quán của dữ liệu sau khi thực hiện, nghĩa là, hoặc tất cả được thực hiện thành công, hoặc không có thao tác nào được thực hiện→ tập hợp các thao tác này được viết thành một transaction.

Ví dụ: Stored procedure thực hiện việc thêm một học sinh vào lớp

```
--Bước 1
Insert  into HocSinh (MaHS, HoTen, MaLop)
values ('hs01','Nguyen V A',1)
--Bước 2
Update Lop
Set SiSo = SiSo +1
```

Nếu bước 2 của stored proc thực hiện không thành công thì dữ liệu trong CSDL có còn nhất quán không?

1.3. Khai báo và sử dụng transaction :

Các lệnh liên quan :

- Bắt đầu transaction :
  - begin tran / begin transaction
- Hoàn tất transaction :
  - commit/ commit tran / commit transaction
- Quay lui transaction :
  - rollback / rollback tran / rollback transaction
- Đánh dấu savepoint trong transaction : save transaction tên_savepoint
- Biến @@trancount : cho biết số transaction hiện đang thực hiện (chưa được kết thúc với rollback hay commit) trong connection hiện hành.

Ghi chú :

- lệnh *rollback tran + tên_savepoint* có tác dụng quay lui giao tác đến vị trí đặt *savepoint* tương ứng (không có tác dụng kết thúc transaction), các khóa (lock) được đặt khi thực hiện các thao tác nằm trong phần bị rollback sẽ được mở ra. *(xem ROLLBACK MỘT PHẦN TRANSACTION)*
- Khi khai báo transaction tường minh, phải đảm bảo rằng sau đó nó được rollback hoặc commit tường minh, nếu không, transaction sẽ tiếp tục tồn tại và chiếm giữ tài nguyên, ngăn trở sự thực hiện của các transaction khác.
- Lệnh rollback chỉ có tác dụng quay lui các **thao tác trên CSDL (thêm, xóa, sửa). Các câu lệnh khác, chẳng hạn lệnh gán, sẽ không bị ảnh hưởng bởi lệnh rollback.**

Các ví dụ : Cho bảng dữ liệu bên dưới. Hãy cho biết giá trị tại các ô màu vàng sau khi thực hiện (độc lập) các transaction trong các ví dụ sau .

MaTS	TuaSach	Tacgia
1	Aaa	ABC
2	Bbb	DEF
3	Ccc	GHI

Vd1 :

	@@trancount = 0
Begin tran --(T1)	@@Trancount = ?
update Tuasach set Tacgia = xxx where MaTS = 1 update TuaSach set TacGia = yyy where MaTS =2	
update TuaSach set TacGia = zzz where MaTS =3	
Commit tran --(T1)	@@trancount =?

Vd 2:

declare @x int set @x=3	@@trancount = 0
Begin tran --(T1)	@@trancount =?
update Tuasach set Tacgia = xxx where MaTS = 1 update TuaSach set TacGia = yyy where MaTS =2	
set @x =7 update TuaSach set TacGia = zzz where MaTS =3	
Rollback --(tran)	@@trancount =? @x =?

Vd 3 :

declare @x int; set @x=3; print @@trancount ;	@@trancount =0
Begin tran ;--(T1)	@@trancount =?

`update Book set Author = 'xxx'  where id = '1';`
`update Book set Author = 'yyy' where id ='2';`
`Save tran S;`
	@@trancount =?

`set @x =7;` `update Book set Author = 'zzz' where id ='3';`	@@trancount =?
`Rollback tran S;`	@@trancount =? @x=?
`Commit tran ;--(T1)`	@@trancount =? @x =?

1.4. Transaction lồng nhau :

- Các transaction có thể thực hiện lồng nhau, mục đích chủ yếu là để cho phép các *stored procedure* có chứa transaction có thể được gọi từ những tiến trình đã nằm bên trong một transaction hoặc từ những tiến trình không nằm bên trong một transaction nào.

  Ví dụ:

```
create proc sp_test
as
      begin tran --T1
            --do something
      commit tran
```

```
begin tran --T2
      exec sp_test
--do something
commit tran
```

→ proc sp_test được gọi từ một tiến trình nằm bên trong một transaction (T2), do đó, transaction T1 trong sp_test được xem là lồng bên trong transaction T2.

→ Chú ý trước câu lệnh gọi *sp_test* SQL Server tự động ghi nhớ giá trị biến hệ thống @@tranCount, giả sử lưu vào biến @trc. Ngay sau câu lệnh này, biến @trc và @@tranCount sẽ được so sánh giá trị, nếu khác nhau thì SQL Server sẽ phát sinh lỗi giúp biết được lệnh gọi stored proc có bảo toàn các cấp lồng nhau về transaction hay không. Đặc biệt khi sp_test gọi rollback thì chắc chắn T2 sẽ có lỗi và toàn bộ transaction đã được rollback; lúc này người dùng được thông báo để tự quyết định thực hiện tiếp hoặc kết thúc tiến trình đang thực hiện.

- SQL đưa ra các qui định sau nhằm đảm bảo việc thực hiện của các transaction lồng nhau không làm vi phạm các tính chất của giao tác :
  - Lệnh commit transaction sẽ được xem như thuộc về transaction bắt đầu sau nhất (bên trong nhất) chưa commit , cho dù nó được đi kèm với tên của transaction bắt đầu trước (cấp ngoài hơn).

- Lệnh commit transaction của transaction con chỉ giảm @@trancount đi 1, không có tác dụng yêu cầu hệ quản trị ghi nhận chắc chắn những thay đổi trên CSDL mà transaction này đã làm.
- Chỉ có lệnh commit transaction của transaction ngoài cùng mới thực sự có tác dụng này (như vậy nếu có n transaction lồng nhau thì lệnh commit transaction thứ n mới thực sự commit toàn bộ giao tác).
- Chỉ cần có một lệnh rollback tran (ở bất cứ cấp nào) là toàn bộ giao tác sẽ bị rollback. Lý do có quy luật này đó là nhằm bảo đảm ý nghĩa *một đơn vị công việc lôgic* bởi khi có lệnh gọi rollback từ một cấp *transaction* nào đó đồng nghĩa với việc không thực hiện thành công tại vị trí đó. Trong một giao tác đang xét, nếu thực hiện không thành công ở bất kỳ điểm nào ở bất kỳ cấp giao tác nào, giao tác ngoài cùng cần được khôi phục bởi đã trong nó đã tồn tại một ví trí "bị lỗi".
- Rollback tran + tran_name chỉ hợp lệ khi "tran_name" là tên của transaction được *save tran* trước đó.
- Giao tác không được lồng nhau quá 32 cấp.
- Các transaction lồng nhau không tranh chấp nhau về tài nguyên (có thể chia sẻ với nhau các khoá trên đơn vị dữ liệu được đọc/ghi). Điều này có thể hiểu từ việc *transaction* lồng nhau chỉ xảy ra trên cùng một *connection*. Mà trên cùng một connection như vậy thì dĩ nhiên không có tranh chấp (bởi các thao tác sẽ được thực hiện tuần tự).

Deleted: ngoài cùng, nếu tran_name là tên của transaction bên trong, lệnh này sẽ bị lỗi

Ghi chú : Nếu có hai transaction T1, T2 thực hiện *trong cùng một connection* thì có hai trường hợp :

- T2 thực hiện sau khi T1 kết thúc, lúc đó T1 và T2 là độc lập nhau;
- T2 thực hiện khi T1 chưa kết thúc, khi đó T2 là giao tác con lồng bên trong T1.

Nói cách khác, không có trường hợp nào xảy ra tranh chấp giữa hai transaction trên cùng connection.

Ví dụ 1:

`declare @x int;` `set @x=3;`	@@trancount =0

`Begin tran ;--(T1)`
 	
`update Book set Author = 'xxx'  where id = '1';`
	@@trancount =?

`update Book set Author = 'yyy' where id ='2';`	

`begin tran;`
 	
`set @x =7;`
`update Book set Author = 'zzz' where id ='3';`
	@@trancount =?

`commit tran ;`	@@trancount =?
`Rollback tran ;--(T1)`	@@trancount =?

Ví dụ 2 :

`declare @x int;`	@@trancount =0

`Begin tran ;--(T1)`
 	
`update Book set Author = 'xxx'  where id = '1';`
`update Book set Author = 'yyy' where id ='2';`
	@@trancount =?

`begin tran T2;`
 	
`update Book set Author = 'zzz' where id ='3';`
	@@trancount =?

`rollback tran T2;`	Lỗi

Ví dụ 3 :

`declare @x int;`	@@trancount =0

`Begin tran ;--(T1)`
 	
`update Book set Author = 'xxx'  where id = '1';`
`update Book set Author = 'yyy' where id ='2';`
	@@trancount =?

`begin tran;`
 	
`update Book set Author = 'zzz' where id ='3';`
	@@trancount =?

`rollback tran;`	@@trancount =?
`Commit/rollback tran;`	Lỗi, tại sao?

1.5. Kiểm tra lỗi khi xây dựng transaction

- Một số lỗi thường gặp sau khi thực hiện 1 câu lệnh trong giao tác:
  - Không có quyền truy cập trên 1 đối tượng (table, stored procedure,…)
  - Vi phạm ràng buộc toàn vẹn (primary key, foreign key, check, rule, các ràng buộc được kiểm tra bằng trigger,…).
  - Deadlock.
  - …
- SQL Server trả giá trị lỗi về trong biến toàn cục @@error.
  - @@error= 0: không xảy ra lỗi
  - @@error <> 0: xảy ra lỗi với mã lỗi là @@error
- **Giao tác không thể tự động rollback khi gặp những lỗi phát sinh trong quá trình thực hiện 1 câu lệnh thành phần trong giao tác**. Vì vậy cần kiểm tra giá trị của biến @@error sau mỗi câu lệnh thành phần trong giao tác và cần xử lý những lỗi (nếu có) và yêu cầu rollback giao tác một cách tường minh bằng lệnh rollback/rollback transaction.

  Ví dụ :
  Xây dựng giao tác thực hiện việc thêm độc giả người lớn (theo đúng mô tả câu 4.9, bài tập quản lý thư viện). Giao tác này nằm trong procedure sp_ThemDocGiaNguoiLon.

```
Create proc sp_ThemDocGiaNguoiLon
@ Ten…
…
as
--buoc 1 : xác định mã độc giả
  declare @madg
  set  @madg = 1
  begin transaction
  while exists (select * from DocGia where ma_docgia = @madg)
        set @madg = @madg +1
   if ( @@error <>0 )
      begin
            rollback tran
            return
       end
-- buoc 2 : insert vao bang docgia
  insert into DocGia    values(…)
  if ( @@error <>0 )
      begin
            rollback tran
            return
       end
-- buoc 3 : kiem tra tuoi
 if  datediff(yy, @ngaysinh, getdate()) <18
 begin
            raiserror('Tuoi  nho hon 18',16,1)
```

```
            rollback tran
            return
end
…
commit transaction
```

2. ROLLBACK MỘT PHẦN TRANSACTION

2.1. Đặt vấn đề :

Thông thường khi chúng ta rollback một transaction thì toàn bộ những thao tác đã thực hiện trong transaction đó đều bị hủy bỏ. Tuy nhiên trong một số trường hợp nhất định chúng ta có nhu cầu chỉ hủy bỏ một số thao tác nào đó mà thôi (các thao tác còn lại không bị hủy bỏ). Tuy nhu cầu này ít khi phát sinh nhưng Microsoft SQLServer có hỗ trợ điều ấy.

2.2. Các câu lệnh và cú pháp

a) Save tran stampName

Lệnh này gắn nhãn tại một vị trí nhất định trong transaction với tên nhãn là *stampName*. Tên nhãn này có thể là một biến chuỗi hay một hằng chuỗi. Khi là hằng chuỗi, tên nhãn **không** được để giữa cặp nháy đơn.

Ví dụ :

```
--Đúng :
Begin tran
…
Save tran nhan_1
…

--Đúng :
Declare @nhan varchar(10)
Set @nhan = 'nhan_2'
Save tran @nhan
…
```

```
--Sai :
Save tran 'nhan_1'
```

b) Rollback tran stampName

Lệnh này rollback một phần của transaction. Các thao tác bị rollback là các thao tác nằm trên câu lệnh "Rollback tran stampName" và nằm dưới câu lệnh "Save tran stampName" với tên nhãn tương ứng.

Ví dụ :

```
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 4 sẽ bị hủy bỏ. Trong khi ấy các thao tác 1 và 2 thì không bị hủy bỏ. Sau đó SQLServer sẽ vẫn tiếp tục làm nốt các thao tác 5 và 6.

Lưu ý : Câu lệnh "rollback tran nhan_1" chỉ đơn thuần làm công việc undo thao tác 3 và 4, nó không hề làm chấm dứt transaction như câu lệnh "rolback tran" thông thường. Nói các khác, sau khi hủy bỏ thao tác 3 và 4, transaction vẫn tiếp tục chạy cho đến khi gặp câu lệnh "rollback tran" hoặc "commit tran" thì mới kết thúc. Trong ví dụ trên, 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.

2.3. Liên hệ các tính chất của transaction

Khi cho transaction rollback một phần, ta cần liên hệ chặt chẽ với các tính chất của transaction như sau :

a) Tính nguyên tố : Việc rollback một phần transaction chắc chắn là vi phạm tính nguyên tố. Nếu sử dụng rollback một phần nghĩa là chấp nhận vi phạm này.

b) Tình vững bền : Việc rollback một phần không liên can gì đến tính vững bền vì nó không làm ảnh hưởng đến phạm vi của transaction. Bất kỳ thao tác nào nằm trong phạm vi transaction mà đã được commit thì sẽ được SQLServer bảo đảm lưu bền vững vào CSDL.

c) Tình nhất quán : Sau khi thực hiện transaction, CSDL vẫn phải ở trong trạng thái nhất quán, nghĩa là không vi phạm các ràng buộc toàn vẹn. Do đó, người viết transaction khi quyết định sử dụng rollback một phần thì phải xem xét kỹ các ràng buộc toàn vẹn để bào đảm dù xảy ra rollback một phần thì cũng không vi phạm ràng buộc toàn vẹn nào.

d) Tính cô lập : Rollback một phần không ảnh hưởng gì đến mức cô lập của transaction nhưng ảnh hưởng rõ rệt đến việc phát khoá và nhả khoá trên các đơn vị dữ liệu. Cơ chế được mô tả trong ví dụ sau :

Ví dụ : Giả sử mức cô lập là Repeatable read

```
[1] Begin tran
[2] Read (A)
[3] Read (B)
[4] Save tran nhan_1
[5] Read (C)
[6] Read (D)
[7] Rollback tran nhan_1
[8] Read (E)
…
```

Sau bước [6], các đơn vị dữ liệu đang bị khóa là : A, B, C và D.

Sau bước [7], các đơn vị dữ liệu đang bị khóa là : A và B. (C và D được khoá trong đoạn mã bị rollback nên sau khi rollback, các khoá này không còn nữa)

Sau bước [8], các đơn vị dữ liệu đang bị khóa là : A, B và E (Có thêm E bị khóa). Các khoá trên A, B và E sẽ được giữ cho đến khi transaction kết thúc.

2.4. Khi nào dùng rollback một phần

2.4.1. Trường hợp 1 - Các thành phần có sự cùng phụ thuộc

Ta thấy rằng hầu hết các transaction đều tuân thủ tính nguyên tố. Việc sử dụng rollback một phần chỉ xảy ra trong những tình huống đặc biệt tùy vào dụng ý và sự linh động của người xây dựng transaction. Ta xét một mẫu transaction như sau :

```
Begin tran                          1
_________
…
Save tran stamp_1                   A
_________
…
If (lỗi 1)
    Rollback tran stamp_1
Save tran stamp_2                   B
_________
…
If (lỗi 2)
    Rollback tran stamp_2
```

```
Save tran stamp_3                   C
_________
…
If (lỗi 3)
    Rollback tran stamp_3
_________                           2
…
Commit tran / Rollback tran
```

Trong mẫu transaction **tổng quát** trên đây, giả sử rằng các đoạn code A, B và C có tính độc lập với nhau. Nghĩa là một trong các đoạn này có thể bị rollback mà không ảnh hưởng đến các đoạn còn lại. Như vậy lẽ ra chúng phải được viết trong 3 transactions khác nhau. Tuy nhiên giả sử rằng các đoạn code A, B và C lại đều phụ thuộc vào đoạn code 1 (ví dụ như đoạn code 1 kiểm tra một ràng buộc nào đó được thỏa thì mới chạy các đoạn code A, B và C). Do đó nếu viết 3 transactions riêng thì phải viết lại đoạn code 1 ba lần.

Hơn thế nữa, ta lại giả sử đoạn code 2 cùng lúc phụ thuộc vào cả 3 đoạn code A, B và C (ví dụ như nó tổng kết những gì làm được và không làm được trong A, B và C). Lúc này, ta không còn giải pháp nào khác là phải đưa cả 5 đoạn code này vào trong 1 transaction. Đây chính là trường hợp ta cần dùng rollback một phần.

Tóm lại : Dùng rollback một phần khi

- Có một nhóm thao tác mà các thao tác trong nhóm này độc lập với nhau (A, B, C) nhưng lại cùng phụ thuộc vào một hay một số thao tác khác ngoài nhóm (1).

- Có một hay một số thao tác nào đó (2) phụ thuộc vào một nhóm thao tác khác mà các thao tác trong nhóm này độc lập nhau (A, B, C).
- Cả hai trường hợp trên.

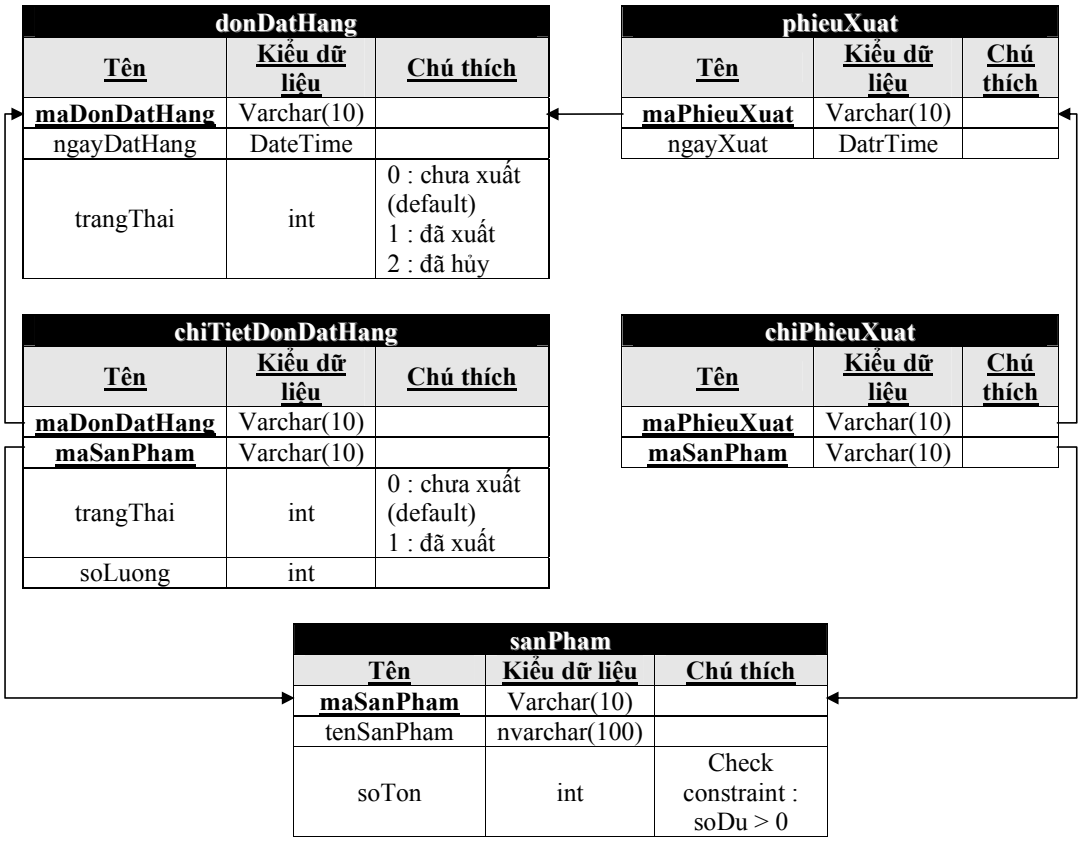
2.4.2. Trường hợp 2 – Kiểm tra ràng buộc tốn kém

Việc rollback một phần thường là giải pháp thay thế cho sự rẽ nhánh logic lúc kiểm tra ràng buộc toàn vẹn (nhánh 1 : Nếu thoả ràng buộc thì làm các thao tác x ; nhánh 2 : Nếu không thỏa thì làm các thao tác y). Giả sử **việc kiểm tra ràng buộc toàn vẹn trong transaction đòi hỏi nhiều tài nguyên** (số dòng trong bảng quá lớn, câu truy vấn phức tạp, …) thì người viết transaction có thể tránh thao tác kiểm tra này bằng cách sử dụng rollback một phần như sau :

- Dùng check constraint, rule hay trigger để ngăn chặn các thay đổi trên dữ liệu làm vi phạm RBTV.
- Trong transaction, xem như không có vi phạm gì xảy ra và cứ thực hiện các thao tác x. Nếu lỡ vi phạm RBTV, các check constraint sẽ phát lỗi (@@error = 547) hoặc các rule sẽ phát lỗi (@@error = 513) hoặc trigger sẽ phát lỗi (mã lỗi do người viết trigger quy định). Lúc đó trong transaction chỉ cần kiểm tra giá trị tương ứng của @@error và cho rollback các thao tác x (chỉ x mà thôi → rollback một phần) rồi thực hiện các thao tác y.

Ví dụ cụ thể

Cho lược đồ cơ sở dữ liệu quản lý đơn đặt hàng như sau :



Stored proc sau đây thực hiện việc xuất hàng cho một đơn đặt hàng đồng thời lập hóa đơn cho việc xuất hàng ấy. Sau khi xuất hàng xong, đơn đặt hàng sẽ có trạng thái là "đã xuất". Với mỗi chi tiết đơn đặt hàng, nếu còn hàng thì xuất và chi tiết ấy được gán trạng thái là "đã xuất" và phát sinh một chi tiết hoá đơn tương ứng. Ngược lại nếu hết hàng rồi thì chi tiết đơn đặt hàng ấy vẫn có trạng thái là "chưa xuất". Nếu tất cả các chi tiết của một đơn đặt hàng đều không có hàng để xuất thì đơn đặt hàng đó có trạng thái là "đã hủy".

```
Create proc xuLyDonDatHang @maDonDatHang varchar(10)
As
If @maDonDatHang = ‘’
Begin
	Print ‘Ma don dat hang phai khac rong’
	Return
End

Begin tran
Set transaction isolation level Serializable
```

```
If not exists(select * from donDatHang where maDonDatHang = @maDonDatHang and
trangThai=0)
Begin
	Print ‘Don dat hang khong ton tai hoac da duoc xu ly roi’
Rollback tran
	Return
End

Insert into phieuXuat values(@maDonDatHang, Getdate())
If (@@error <> 0)
Begin
	Print ‘Khong the them phieu xuat’
	Rollback tran
	return
End
Declare cur_chiTiet cursor for (select maSanPham, soLuong from chiTietDonDatHang where
maDondatHang = @maDonDatHang)
Open cur_chiTiet
Declare @maSanPham vachar(10)
Declare @soLuong int
Declare @soChiTietXuat int
Set @soChiTietXuat = 0

Fetch next from cur_chiTiet into @maSanPham, @soLuong
While (@@ferch_status = 0)
Begin
	Save tran @maSanPham
Insert into chiTietPhieuXuat (@maDonDatHang,@maSanPham)
If (@@error <> 0)
Begin
		Print ‘Khong the them phieu xuat’
		Rollback tran
		return
End

Update chiTietDonDatHang set trangThai = 1 where maDonDatHang = @maDonDatHang and
maSanPham = @maSanPham
If (@@error <> 0)
Begin
		Print ‘Khong cap nhat duoc trang thai chi tiet don dat hang’
		Rollback tran
		return
End

	Update sanPham set soTon = soTon - @soLuong where maSanPham = @maSanPham
If (@@error = 547)
		Rollback tran @maSanPham
	ElseIf (@@error <> 0)
Begin
		Print ‘Khong the them phieu xuat’
		Rollback tran
		return
End
Else
	Set @soChiTietXuat = @soChiTietXuat + 1
```

```
Fetch next from cur_chiTiet into @maSanPham, @soLuong
End
If (@soChiTietXuat = 0)
Begin
	Print ‘Toan bo don dat hang bi huy’
	Update donDatHang set trangThai = 2 where maDondatHang = @maDondatHang
	Rollback tran
	Return
End
Update donDatHang set trangThai = 1 where maDondatHang = @maDondatHang
If (@@error <> 0)
Begin
	Print ‘Khong cap nhat duoc trang thai don dat hang’
	Rollback tran
	return
End
Commit tran
Go
```