```
--Batch
--1) Viết một batch khai báo biến @tongsoHD chứa tổng số hóa đơn của sản phẩm
--có ProductID='778', nếu @tongsoHD>500 thì in ra chuỗi "San pham 778 có
--trên 500 đơn hàng", ngược lại tin ra chuỗi "San pham 778 co it don dat hang"

declare @tongsoHD int, @masp int
set @masp=778
set @tongsoHD=(select count(*)
                  from [Sales].[SalesOrderDetail]
                  where [ProductID]=@masp
                  )
if @tongsoHD>500
     print N'Sản phẩm ' +cast(@masp as char(3))+ N' có trên 500 đơn hàng'
else
     print N'Sản phẩm ' +cast(@masp as char(3))+ N' có ít đơn hàng'


select [SalesOrderID], [ProductID]
from [Sales].[SalesOrderDetail]
order by [ProductID]
--2) Viết một đoạn Batch với tham số @makh và @n chứa số hóa đơn của khách
--hàng @makh, tham số @nam chứa năm lập hóa đơn (ví dụ @nam=2008), nếu
--@n>0 thì in ra chuỗi:"Khách hàng có @n hóa đơn trong năm 2008" ngược lại
--nếu @n=0 thì in ra chuỗi "Khách hàng không có hóa đơn nào trong năm 2008"

declare @makh int, @n int, @nam int
set @nam=2008
set @makh=11000
set @n=(select count(*)
          from [Sales].[SalesOrderHeader]
          where [CustomerID]=@makh and YEAR([OrderDate])=@nam
          )
if @n>0
```

```
      print N'Khách hàng' + cast(@makh as char (5))+N' có'+ cast(@n as char(4)) + N'hóa đơn trong
năm'+ cast(@nam as char(4))
else
      print N'Khách hàng' + cast(@makh as char (5))+N' không có hóa đơn trong năm'+ cast(@nam as
char(4))
---
--Viết một batch tính số tiền giảm cho những hóa đơn (SalesOrderID) có tổng
--tiền>100000, thông tin gồm [SalesOrderID], Subtotal=sum([LineTotal]),
--Discount (tiền giảm), với Discount được tính như sau:
--Những hóa đơn có Subtotal<100000 thí không giảm,
--Subtotal từ 100000 đến <120000 thì giảm 5% của Subtotal
--Subtotal từ 120000 đến <150000 thì giảm 10% của Subtotal
--Subtotal từ 150000 trở lên thì giảm 15% của Subtotal

 select h.SalesOrderID, Subtotal=sum([LineTotal]), Discount= (
                case
                      when sum([LineTotal])<100000 then 0
                      when sum([LineTotal]) between 100000 and 120000 then sum([LineTotal])*0.05
                      when sum([LineTotal]) between 120000 and 150000 then sum([LineTotal])*0.1
                      else sum([LineTotal])*0.15
                end )

 from [Sales].[SalesOrderHeader] h join [Sales].[SalesOrderDetail] d
 on h.SalesOrderID=d.SalesOrderID
 group by h.SalesOrderID
 having sum([LineTotal])>100000

 select h.SalesOrderID, Subtotal=sum([LineTotal])
 from [Sales].[SalesOrderHeader] h join [Sales].[SalesOrderDetail] d
 on h.SalesOrderID=d.SalesOrderID
 group by h.SalesOrderID
 having sum([LineTotal])>100000
```

```
--Moduel1-----------------------------------------------
```

```sql
--cau 3
CREATE DATABASE SmallWorks
ON
PRIMARY
(
      NAME = 'SmallWorksPrimary',
      FILENAME = 'T:\data\SmallWorks.mdf',
      SIZE = 10MB,
      FILEGROWTH = 20%,
      MAXSIZE = 50MB
),
FILEGROUP SWUserData1
(
      NAME = 'SmallWorksData1',
      FILENAME = 'T:\data\SmallWorksData1.ndf',
      SIZE = 10MB,
      FILEGROWTH = 20%,
      MAXSIZE = 50MB
),
FILEGROUP SWUserData2
(
      NAME = 'SmallWorksData2',
      FILENAME = 'T:\data\SmallWorksData2.ndf',
      SIZE = 10MB,
      FILEGROWTH = 20%,
      MAXSIZE = 50MB
)
LOG ON
(
      NAME = 'SmallWorks_log',
      FILENAME = 'T:\data\SmallWorks_log.ldf',
      SIZE = 10MB,
      FILEGROWTH = 10%,
```

```
      MAXSIZE = 20MB
)
--CAU 5
--TAO THEM FILEGROUP
      ALTER DATABASE SmallWorks
      add FILEGROUP Test1FG1
-- ADD FILE VAO FILE GROUP
      ALTER DATABASE SmallWorks
      ADD File
      (
            NAME=' filedat1',
            FILENAME = 't:\data\filedat1.ndf',
            SIZE = 5MB,
            FILEGROWTH = 20%,
            MAXSIZE = 50MB
      ) ,
      (
            NAME=' filedat2',
            FILENAME = 't:\data\filedat2.ndf',
            SIZE = 10MB,
            FILEGROWTH = 20%,
            MAXSIZE = 50MB
      )
      to FILEGROUP  Test1FG1
--CAU 6
ALTER DATABASE SmallWorks
ADD FILE
(
      NAME=' filedat3',
      FILENAME = 't:\data\filedat3.ndf',
      SIZE = 3MB,
      FILEGROWTH = 20%,
      MAXSIZE = 50MB
```

```
)
to FILEGROUP  Test1FG1
---
ALTER DATABASE SmallWorks
MODIFY FILE
(
      NAME = 't:\data\filedat3.ndf',
      SIZE = 5M
)
--cau 7: xao file group (file phai rong)

ALTER DATABASE SmallWorks
REMOVE FILEGROUP Test1FG1

--cau 8: xem thuoc tinh cua CSDL
 sp_helpDb SmallWorks
 sp_spaceused
 sp_helpfile
--cau 9
      ALTER DATABASE SmallWorks
      MODIFY FILEGROUP Test1FG1 READONLY
--cau 10
CREATE TABLE dbo.Person
(
      PersonID int NOT NULL,
      FirstName varchar(50) NOT NULL,
      MiddleName varchar(50) NULL,
      LastName varchar(50) NOT NULL,
      EmailAddress nvarchar(50) NULL
)

select [BusinessEntityID],[FirstName] , MiddleName, LastName, [EmailPromotion]
into dbo.Person
```

```
from AdventureWorks2008R2.Person.Person
 select*from dbo.Person
 --into <ten bang moi>
--from <AdventureWorks2008R2.>
```

```
--Module2
--1) Liệt kê danh sách các hóa đơn (SalesOrderID) lập trong tháng 6 năm 2008 có tổng tiền >70000,
--thông tin gồm SalesOrderID, Orderdate, SubTotal, trong đó SubTotal =sum(OrderQty*UnitPrice).
     select d.SalesOrderID, OrderDate, SubTotal=sum(OrderQty * UnitPrice)
     from sales.SalesOrderDetail d join Sales.SalesOrderHeader h on d.SalesOrderID =
h.SalesOrderID
     where  MONTH(OrderDate) = 6 and YEAR(OrderDate) = 2008
     group by d.SalesOrderID, OrderDate
     having SUM(OrderQty * UnitPrice) > 70000

--2) Đếm tổng số khách hàng và tổng tiền của những khách hàng thuộc các quốc gia có mã vùng là US
--(lấy thông tin từ các bảng SalesTerritory, Sales.Customer, Sales.SalesOrderHeader,
Sales.SalesOrderDetail).
--Thông tin bao gồm TerritoryID, tổng số khách hàng (countofCus), tổng tiền (Subtotal) với
Subtotal = SUM(OrderQty*UnitPrice)

     select t.TerritoryID, CountofCus= COUNT(c.CustomerID) , Subtotal=SUM(d.OrderQty *
d.UnitPrice)
     from Sales.SalesTerritory t join Sales.Customer  c on t.TerritoryID=c.TerritoryID
                                       join Sales.SalesOrderHeader h on
h.CustomerID=h.CustomerID
                                       join Sales.SalesOrderDetail d on
h.SalesOrderID=d.SalesOrderID
     where CountryRegionCode = 'US'
     group by t.TerritoryID


--3)Tính tổng trị giá của những hóa đơn với Mã theo dõi giao hàng (CarrierTrackingNumber) có 3 ký
tự đầu là 4BD,
```

```
-- thông tin bao gồm SalesOrderID, CarrierTrackingNumber, SubTotal=sum(OrderQty*UnitPrice)

     select SalesOrderID, CarrierTrackingNumber, Subtotal=SUM(OrderQty * UnitPrice)
     from Sales.SalesOrderDetail
     where CarrierTrackingNumber like '4BD%'
     group by SalesOrderID, CarrierTrackingNumber

--4) Liệt kê các sản phẩm (product)có đơn giá (unitPrice)<25 và số lượng bán trung bình >5, thông
tin gồm ProductID, name,
--AverageofQty

     select pro.ProductID, pro.Name, AverageofQty=AVG(det.OrderQty)
     from Sales.SalesOrderDetail det join Production.Product pro on det.ProductID = pro.ProductID
     where det.UnitPrice < 25
     group by pro.ProductID, pro.Name
     having AVG(det.OrderQty) > 5

--5) Liệt kê các công việc (JobTitle) có tổng số nhân viên >20 người, thông tin gồm JobTitle,
countofPerson=count(*)

     select JobTitle, CountofEmployee=count(BusinessEntityID)
     from HumanResources.Employee
     group by JobTitle
     having COUNT(BusinessEntityID) > 20

--6) Tính tổng số lượng và tổng trị giá của các sản phẩm do các nhà cung cấp có tên kết thúc bằng
‘Bicycles’ và tổng trị giá >800000,
--   thông tin gồm BusinessEntityID, Vendor_name, ProductID, sumofQty, SubTotal (sử dụng các bảng
[Purchasing].[Vendor] [Purchasing].
--   [PurchaseOrderHeader] và [Purchasing].[PurchaseOrderDetail])

     select v.BusinessEntityID, v.Name, ProductID, sumofQty = SUM(OrderQty), SubTotal =
SUM(OrderQty * UnitPrice)
```

```
	from Purchasing.Vendor v join Purchasing.PurchaseOrderHeader h on h.VendorID =
v.BusinessEntityID
                                        join Purchasing.PurchaseOrderDetail d on h.PurchaseOrderID
= d.PurchaseOrderID
	where v.Name like '%Bicycles'
	group by v.BusinessEntityID, v.Name, ProductID
	having SUM(OrderQty * UnitPrice) > 800000


--7) Liệt kê các sản phẩm có trên 500 đơn đặt hàng trong quí 1 năm 2008 và có tổng trị giá >10000,
thông tin gồm ProductID,
--Product_name, countofOrderID và Subtotal
	select p.ProductID, p.Name, countofOrderID = COUNT(o.SalesOrderID), Subtotal = sum(OrderQty
* UnitPrice)
	from Production.Product p join Sales.SalesOrderDetail o on p.ProductID = o.ProductID
                                        join sales.SalesOrderHeader h on h.SalesOrderID =
o.SalesOrderID
	where Datepart(q, OrderDate) =1 and YEAR(OrderDate) = 2008
	group by p.ProductID, p.Name
	having sum(OrderQty * UnitPrice) > 10000 and COUNT(o.SalesOrderID) > 500

--8) Liệt kê danh sách các khách hàng có trên 25 hóa đơn đặt hàng từ năm 2007 đến 2008,
--thông tin gồm mã khách (PersonID) , họ tên (FirstName +' '+ LastName as fullname), Số hóa đơn
(CountOfOrders).

select PersonID, FirstName +' '+ LastName as fullname, CountOfOrders=count(*)
from [Person].[Person] p join [Sales].[Customer] c on p.BusinessEntityID=c.CustomerID
                                    join [Sales].[SalesOrderHeader] h on h.CustomerID= c.CustomerID
where YEAR([OrderDate])>=2007 and YEAR([OrderDate])<=2008
group by PersonID, FirstName +' '+ LastName
having count(*)>25
```

```sql
--9) Liệt kê những sản phẩm có tên bắt đầu với 'Bike' và 'Sport' có tổng số lượng bán trong mỗi
mỗi năm trên 500 sản phẩm,
--thông tin gồm ProductID, Name, CountofOrderQty, year. (dữ liệu lấy từ các bảng
Sales.SalesOrderHeader, Sales.SalesOrderDetail,
-- and Production.Product)
select p.ProductID, Name, CountofOrderQty=sum([OrderQty]), yearofSale=year([OrderDate])
from [Production].[Product] p join [Sales].[SalesOrderDetail] d on p.ProductID=d.ProductID
                                        join [Sales].[SalesOrderHeader] h on
d.SalesOrderID=d.SalesOrderID
where name like 'Bike%' or name like 'Sport%'
group by p.ProductID, Name, year([OrderDate])
having sum([OrderQty])>500

--10) Liệt kê những phòng ban có lương (Rate: lương theo giờ) trung bình >30,
-- thông tin gồm Mã phòng ban (DepartmentID), tên phòng ban (name), Lương trung bình (AvgofRate).
-- Dữ liệu từ các bảng [HumanResources].[Department],
[HumanResources].[EmployeeDepartmentHistory],
-- [HumanResources].[EmployeePayHistory]
select d.DepartmentID, d.name, AvgofRate=avg([Rate])
from [HumanResources].[Department] d join [HumanResources].[EmployeeDepartmentHistory] h on
d.DepartmentID=h.DepartmentID
                                                join [HumanResources].[EmployeePayHistory] e on
h.BusinessEntityID=e.BusinessEntityID
group by d.DepartmentID, d.name
having avg([Rate])>30
```

```sql
--Liệt kê các sản phẩm gồm các thông tin product names và product ID có trên
--100 đơn đặt hàng trong tháng 7 năm 2008

select ProductID, Name
from Production.Product
where ProductID in (select ProductID
                            from  Sales.SalesOrderDetail d join Sales.SalesOrderHeader h on
d.SalesOrderID=h.SalesOrderID
```

```
                            where MONTH(OrderDate)=7 and YEAR(OrderDate)=2008
                            group by  ProductID
                            having COUNT(*)>100)
---
select ProductID, Name
from Production.Product p
where  exists (select ProductID
                            from  Sales.SalesOrderDetail d join Sales.SalesOrderHeader h on
d.SalesOrderID=h.SalesOrderID
                            where MONTH(OrderDate)=7 and YEAR(OrderDate)=2008 and
ProductID=p.ProductID
                            group by  ProductID
                            having COUNT(*)>100)
----cau 2
--Liệt kê các sản phẩm (ProductID, name) có số hóa đơn đặt hàng nhiều nhất trong
--tháng 7/2008
select p.ProductID, Name
from Production.Product p join Sales.SalesOrderDetail d on p.ProductID=d.ProductID
                            join Sales.SalesOrderHeader h on d.SalesOrderID=h.SalesOrderID
where  MONTH(OrderDate)=7 and YEAR(OrderDate)=2008
group by p.ProductID, Name
having COUNT(*)>=all( select COUNT(*)
                            from Sales.SalesOrderDetail d join Sales.SalesOrderHeader h on
d.SalesOrderID=h.SalesOrderID
                        where MONTH(OrderDate)=7 and YEAR(OrderDate)=2008
                            group by ProductID
                            )
---cau 3
select [CustomerID], count(*)
from [Sales].[SalesOrderHeader]
group by [CustomerID]
having count(*)>=all( select count(*)
                                from [Sales].[SalesOrderHeader]
```

```
                                group by [CustomerID]
                            )
--Hiển thị thông tin của khách hàng có số đơn đặt hàng nhiều nhất, thông tin gồm:
--CustomerID, Name, CountofOrder
select     c.CustomerID, CountofOrder=COUNT(*)
from Sales.Customer c join Sales.SalesOrderHeader h on c.CustomerID=h.CustomerID
group by c.CustomerID
having COUNT(*)>=all(select COUNT(*)
                            from Sales.Customer c join Sales.SalesOrderHeader h on
c.CustomerID=h.CustomerID
                            group by c.CustomerID)
----cau 4
--Liệt kê các sản phẩm (ProductID, Name) thuộc mô hình sản phẩm áo dài tay với
--tên bắt đầu với “Long-Sleeve Logo Jersey”, dùng phép IN và EXISTS, (sử dụng
--bảng Production.Product và Production.ProductModel
select ProductID, Name
from Production.Product
where ProductModelID in (select ProductModelID
                                from Production.ProductModel
                                where Name like 'Long-Sleeve Logo Jersey%')

select ProductID, Name
from Production.Product p
where exists (select ProductModelID
                                from Production.ProductModel
                                where Name like 'Long-Sleeve Logo Jersey%' and
ProductModelID=p.ProductModelID)
--cau 5
--Tìm các mau sản phẩm (ProductModelID) mà giá niêm yết (list price) tối đa
--cao hơn giá trung bình của tất cả các mô hình.
select p.ProductModelID, m.Name, max(ListPrice)
from Production.ProductModel m join Production.Product p on m.ProductModelID=p.ProductModelID
group by p.ProductModelID, m.Name
```

```sql
having max(ListPrice)>=all(select AVG(ListPrice)
                                    from Production.ProductModel m join Production.Product p on
m.ProductModelID=p.ProductModelID
                                    )

--cau 6
--Liệt kê các sản phẩm gồm các thông tin ProductID, Name, có tổng số lượng đặt
--hàng >5000 (dùng In, exists)
select ProductID, Name
from Production.Product
where ProductID in (select ProductID
                           from Sales.SalesOrderDetail
                           group by ProductID
                           having SUM(OrderQty)>5000)
select ProductID, Name
from Production.Product p
where exists (select ProductID
                           from Sales.SalesOrderDetail
                           where ProductID=p.ProductID
                           group by ProductID
                           having SUM(OrderQty)>5000)
--cau 7
--Liệt kê những sản phẩm (ProductID, UnitPrice) có đơn giá (UnitPrice) cao nhất
--trong bảng Sales.SalesOrderDetail
select distinct ProductID, UnitPrice
from Sales.SalesOrderDetail
where UnitPrice>=all (select distinct UnitPrice
                            from Sales.SalesOrderDetail)

--cau 8
--Liệt kê các sản phẩm không có đơn đặt hàng nào thông tin gồm ProductID,
--Name, dùng 3 cách Not in, not exists và left join
select P.productID, Name
```

```
from Production.Product p left join Sales.SalesOrderDetail d on p.ProductID=d.ProductID
where d.ProductID is null

select productID, Name
from Production.Product
where productID not in (select productID
                                from Sales.SalesOrderDetail)

select productID, Name
from Production.Product p
where not exists (select productID
                        from Sales.SalesOrderDetail
                        where p.ProductID=ProductID)
---cau 9
--Liệt kê các nhân viên không lập hóa đơn từ sau ngày 1/5/2008, thông tin gồm
--EmployeeID, FirstName, LastName (dữ liệu từ 2 bảng HR.Employees và
--Sales.Orders)
select [BusinessEntityID] as EmployeeID, FirstName, LastName
from [Person].[Person]
where [BusinessEntityID]  in (select [SalesPersonID]
                                            from [Sales].[SalesOrderHeader]
                                            where [OrderDate]>'2008-5-1'
                                            )

--cau 10. Liệt kê danh sách các khách hàng (customerID, name) có hóa đơn đặt hàng trong
--năm 2007 nhưng có hóa đơn đặt hàng trong năm 2008.
select [CustomerID]
from [Sales].[SalesOrderHeader]
where [CustomerID] in (select [CustomerID]
                               from [Sales].[SalesOrderHeader]
                               where year([OrderDate])=2007 )
     and [CustomerID] not in (select [CustomerID]
                               from [Sales].[SalesOrderHeader]
```

```
                                    where year([OrderDate])=2008)
```

```
--1) Tạo hai bảng mới trong cơ sở dữ liệu AdventureWorks2008 theo cấu trúc sau:
create table MyDepartment
(
      DepID smallint not null primary key,
      DepName nvarchar(50),
      GrpName nvarchar(50)
)
create table MyEmployee (
      EmpID int not null primary key,
      FrstName nvarchar(50),
      MidName nvarchar(50),
      LstName nvarchar(50),
      --DepID smallint not null foreign key references
      --MyDepartment(DepID)
)
--2) Dùng lệnh insert <tableName1> select <fieldList>from <TableName2>
--chèn dữ liệu cho bảng MyDepartment, lấy dữ liệu từ bảng
--[HumanResources].[Department].
      insert MyDepartment
      select [DepartmentID], [Name], [GroupName]
      from [HumanResources].[Department]
      select*from MyDepartment
--3) Tương tự câu 2, chèn 20 dòng dữ liệu cho bảng MyEmployee lấy dữ liệu từ 2
--bảng
--[Person].[Person]và
--[HumanResources].[EmployeeDepartmentHistory]
      insert MyEmployee
      select top 20 [BusinessEntityID], [FirstName], [MiddleName],[LastName], null
      from [Person].[Person]
      where [BusinessEntityID]<=20
```

```
alter table MyEmployee
add constraint fk_DepID foreign key (DepID) references MyDepartment (DepID)

select*from MyEmployee
delete from MyDepartment
where DepID=1--khong xoa duoc
----
alter table MyEmployee
add constraint df_DepID default 1 for DepID
---
insert into MyEmployee (EmpID, FrstName, MidName, LstName)
values(21, 'Nguyen','Nhat','Nam')
---
alter table MyEmployee
add constraint fk_DepID foreign key (DepID) references MyDepartment (DepID)
on delete set default
---
delete from MyDepartment
where DepID=1
---
alter table MyEmployee
drop constraint fk_DepID
----
alter table MyEmployee
add constraint fk_DepID foreign key (DepID) references MyDepartment (DepID)
on update cascade
----
alter table MyEmployee
drop constraint fk_DepID
---
alter table MyEmployee
add constraint fk_DepID foreign key (DepID) references MyDepartment (DepID)
on delete cascade --xoa record trong bang cha thi cac record trong bang con bi xoa
```

```
delete from
--4) Dùng lệnh delete xóa 1 record trong bảng MyDepartment với DepID=1, có thực
--hiện được không? Vì sao?
--5) Thêm một default constraint vào field DepID trong bảng MyEmployee, với
--giá trị mặc định là 1.
--Mục tiêu:
--  Thực hiện các ràng buộc toàn vẹn dữ liệu: Primary key, foreign key, domain,
--check, default.
--  Tìm hiểu cascading constraint trong thao tác update và delete6) Nhập thêm một record mới trong
bảng MyEmployee, theo cú pháp sau:
--insert into MyEmployee (EmpID, FrstName, MidName,
--LstName) values(1, 'Nguyen',’Nhat’,'Nam'), quan sát giá trị
--trong field depID của record mới thêm.
--7) Xóa foreign key constraint trong bảng MyEmployee, thiết lập lại khóa ngoại
--DepID tham chiếu đến DepID của bảng MyDepartment với thuộc tính on delete
--set default.
--8) Xóa một record trong bảng MyDepartment có DepID=7, quan sát kết quả trong
--hai bảng MyEmployee và Mydepartment
--9) Xóa foreign key trong bảng MyEmployee hiệu chỉnh ràng buộc khóa ngoại
--DepID trong bảng MyEmployee, thiết lập thuộc tính on delete cascading và on
--update cascading
--10)Thực hiện xóa một record trong bảng MyDepartment với DepID =3, có thực
--hiện được không?
--11)Thêm ràng buộc check vào bảng MyDepartment tại field GrpName, chỉ cho phép
--nhận thêm những Department thuộc group Manufacturing
--12)Thêm ràng buộc check vào bảng [HumanResources].[Employee], tại cột
--Birthday, chỉ cho phép nhập thêm nhân viên mới có tuổi từ 18 đến 60
```

```
--1.Tạo view dbo.vw_Products hiển thị danh sách các sản phẩm từ bảng
--Production.Product và bảng Production.ProductCostHistory. Thông tin bao gồm
--ProductID, Name, Color, Size, Style, StandardCost, EndDate, StartDate
go
create view vw_Products1
```

```
as
	select p.ProductID, Name, Color, Size, Style, p.StandardCost, EndDate, StartDate
	from [Production].[Product] p join [Production].[ProductCostHistory] c
	on p.ProductID=c.ProductID

go
select*from [dbo].[vw_Products1]
sp_helptext [vw_Products1]
--2) Tạo view List_Product_view chứa danh sách các sản phẩm có trên 500 đơn đặt
--hàng trong quí 1 năm 2008 và có tổng trị giá >10000, thông tin gồm ProductID,
--Product_name, countofOrderID và Subtotal.
go
create view List_Product_view
as
	select p.[ProductID], [Name] as Product_name, countofOrderID= count(*),
Subtotal=sum([OrderQty]*[UnitPrice])
	from [Production].[Product] p join  [Sales].[SalesOrderDetail] o on p.ProductID=o.ProductID
                                        join [Sales].[SalesOrderHeader] h on
o.SalesOrderID=h.SalesOrderID
	where datepart(q, [OrderDate])=1 and YEAR([OrderDate])=2008
	group by p.[ProductID], [Name]
	having sum([OrderQty]*[UnitPrice])>10000 and count(*)>500


--3) Tạo view dbo.vw_CustomerTotals hiển thị tổng tiền bán được (total sales) từ cột
--TotalDue của mỗi khách hàng (customer) theo tháng và theo năm. Thông tin gồm
--CustomerID, YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS
--OrderMonth, SUM(TotalDue).
go
create view vw_CustomerTotals
as
	select CustomerID, YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS OrderMonth,
sumofTotal=SUM(TotalDue)
```

```
	from [Sales].[SalesOrderHeader]
	group by CustomerID, YEAR(OrderDate), MONTH(OrderDate)
--4) Tạo view trả về tổng số lượng sản phẩm (total quantity) bán được của mỗi nhân
--viên theo từng năm. Thông tin gồm SalesPersonID, OrderYear, sumOfOrderQty
go
create view view_SumofQty
as
	select SalesPersonID, OrderYear=year([OrderDate]), sumOfOrderQty=sum([OrderQty])
	from [Sales].[SalesOrderHeader] h join [Sales].[SalesOrderDetail] d on
h.SalesOrderID=d.SalesOrderID
	group by SalesPersonID, year([OrderDate])
--5) Tạo view ListCustomer_view chứa danh sách các khách hàng có trên 25 hóa đơn
--đặt hàng từ năm 2007 đến 2008, thông tin gồm mã khách (PersonID) , họ tên
--(FirstName +' '+ LastName as fullname), Số hóa đơn (CountOfOrders).
go
create view ListCustomer_view
as
	select  [CustomerID], FirstName +' '+ LastName as fullname
	from [Sales].[SalesOrderHeader] h join [Person].[Person] p on
h.CustomerID=p.BusinessEntityID
	where year([OrderDate])>=2007 AND year([OrderDate])<=2008
	GROUP BY  [CustomerID], FirstName +' '+ LastName
	HAVING count(*)>25

--6) Tạo view ListProduct_view chứa danh sách những sản phẩm có tên bắt đầu với
--‘Bike’ và ‘Sport’ có tổng số lượng bán trong mỗi mỗi năm trên 500 sản phẩm,
--thông tin gồm ProductID, Name, CountofOrderQty, year. (dữ liệu lấy từ các
--bảng Sales.SalesOrderHeader, Sales.SalesOrderDetail, and
--Production.Product)
go
create view ListProduct_view
as
	select p.ProductID, Name, SumofOrderQty=sum([OrderQty]), year([OrderDate]) as yearofOrder
```

```
	from [Production].[Product] p join [Sales].[SalesOrderDetail] d on p.ProductID=d.ProductID
                                                join[Sales].[SalesOrderHeader] h on
d.SalesOrderID=h.SalesOrderID
	where Name like 'Bike%' or Name like 'Sport%'
	group by p.ProductID, Name, year([OrderDate])
	having sum([OrderQty])>500
--7) Tạo view List_department_View chứa danh sách các phòng ban có lương (Rate:
--lương theo giờ) trung bình >30, thông tin gồm Mã phòng ban (DepartmentID),
--tên phòng ban (name), Lương trung bình (name). Dữ liệu từ các bảng
--[HumanResources].[Department],
--[HumanResources].[EmployeeDepartmentHistory],
--[HumanResources].[EmployeePayHistory].
go
create view List_department_View
as
	select d.DepartmentID, name, avgofRate=avg(Rate)
	from [HumanResources].[Department] d join [HumanResources].[EmployeeDepartmentHistory] e on
d.DepartmentID=e.DepartmentID
                                                        join [HumanResources].[EmployeePayHistory]
h on e.BusinessEntityID=h.BusinessEntityID
	group by d.DepartmentID, name
	having avg(Rate)>30
--8) Tạo view Sales.vw_OrderSummary với từ khóa WITH ENCRYPTION gồm
--orderYear (năm của ngày lập), OrderMonth (tháng của ngày lập), OrderTotal
--(tổng tiền). Sau đó xem thông tin và trợ giúp về mã lệnh của view này
go
create view vw_OrderSummary WITH ENCRYPTION
as
	select year([OrderDate]) as orderYear, month([OrderDate]) as OrderMonth,
OrderTotal=sum([OrderQty]*[UnitPrice])
	from [Sales].[SalesOrderHeader] h join [Sales].[SalesOrderDetail] d on
h.SalesOrderID=h.SalesOrderID
	group by year([OrderDate]), month([OrderDate])
```

```
go
sp_helptext [List_Product_view]
sp_helptext vw_OrderSummary
--9) Tạo view Production.vwProducts với từ khóa WITH SCHEMABINDING
--gồm ProductID, Name, StartDate,EndDate,ListPrice của bảng Product và bảng
--ProductCostHistory. Xem thông tin của View. Xóa cột ListPrice của bảng
--Product. Có xóa được không? Vì sao?
go
create view vwProducts WITH SCHEMABINDING
as
     select p.ProductID, Name, StartDate,EndDate,ListPrice
     from  [Production].[Product] p join  [Production].[ProductCostHistory] h on
p.ProductID=h.ProductID

--10)Tạo view view_Department với từ khóa WITH CHECK OPTION chỉ chứa các
--phòng thuộc nhóm có tên (GroupName) là “Manufacturing” và “Quality
--Assurance”, thông tin gồm: DepartmentID, Name, GroupName.
go
create view view_Department
as
     select DepartmentID, Name, GroupName
     from [HumanResources].[Department]
     where GroupName='Manufacturing' or GroupName='Quality Assurance'
     WITH CHECK OPTION
--a. Chèn thêm một phòng ban mới thuộc nhóm không thuộc hai nhóm
--“Manufacturing” và “Quality Assurance” thông qua view vừa tạo. Có
--chèn được không? Giải thích
go
insert view_Department values( 'nhan su', 'a')

select *from [HumanResources].[Department]
--b. Chèn thêm một phòng mới thuộc nhóm “Manufacturing” và một phòng
--thuộc nhóm “Quality Assurance”.
```

```
insert view_Department values( 'nhan su', 'Manufacturing')
--c. Dùng câu lệnh select xem kết quả trong bảng Department
```

```
--==========Module6: Trigger
--cau 1============================================================
--+Tạo một Instead of trigger thực hiện trên view
--+Tạo một view tên EmpDepart_view bao gồm các field: EmployeeID, Firtname,
-- MiddleName, LastName, e.DepartmentID, Name, groupName, dựa trên 2 bảng
-- M_employees và M_Department.
--+Tạo một trigger tên InsteadOf_Trigger thực hiện trên view EmpDepart_view,
-- dùng để chèn dữ liệu vào các bảng M_employees và M_Department khi chèn
-- một record mới thông qua view EmpDepart_view.
--====================================================================
create table M_Department
(
	DepartmentID int not null primary key,
	Name nvarchar(50),
	groupName nvarchar(50)
)
create table M_Employees
(
	EmployeeID int not null primary key,
	Firtname nvarchar(50),
	MiddleName nvarchar(50),
	LastName nvarchar(50),
	DepartmentID int foreign key references M_Department(DepartmentID)
```

```
)
--tạo view
      go
      create view view_trigger
      as
            select EmployeeID, Firtname, MiddleName, LastName, e.DepartmentID, Name, groupName
            from M_Employees e join M_Department d on e.DepartmentID=d.DepartmentID
      go
---tạo trigger
      create trigger insteadof_trigger on view_trigger
      instead of insert
      as
            begin
                  insert M_Department
                  select DepartmentID, Name, groupName from inserted
                  insert M_Employees
                  select EmployeeID, Firtname, MiddleName, LastName, DepartmentID
                  from inserted
            end
--Test triiger
      select*from view_trigger
      insert view_trigger values(1, 'Nguyen','Hoang','Huy', 11,'Marketing','Sales')
      select*from M_Department
      select*from M_Employees
--cau 2============================================================
--Tạo một trigger thực hiện trên bảng MySalesOrders có chức năng thiết lập độ ưu
--tiên của khách hàng (custpriority) khi người dùng thực hiện các thao tác Insert,
--Update và Delete trên bảng MySalesOrders theo điều kiện như sau:
--▯ Nếu tổng tiền Sum(SubTotal) của khách hàng dưới 10,000 $ thì độ ưu tiên của
--khách hàng (custpriority) là 3
--▯ Nếu tổng tiền Sum(SubTotal) của khách hàng từ 10,000 $ đến dưới 50000 thì
--độ ưu tiên của khách hàng (custpriority) là 2
--▯ Nếu tổng tiền Sum(SubTotal) của khách hàng từ 50000$ trở lên thì độ ưu tiên
```

```
--của khách hàng (custpriority) là 1
--================================================================
--Tạo 2 bảng dữ liệu, chèn dữ liệu lấy từ các bảng trong AdventureWorks
create table Mcustomer
(
	customerID int primary key,
	custpriority int
)
insert Mcustomer ([CustomerID],custpriority)
select [CustomerID], null
from [Sales].[Customer]
where CustomerID>30100 and CustomerID<30118
select*from Mcustomer

create table MsalesOrders
(
	SalesOrderID int primary key,
	OrderDate date,
	SubTotal money,
	customerID int foreign key references Mcustomer(customerID)
)

insert MsalesOrders
select [SalesOrderID],OrderDate, [SubTotal], [CustomerID]
from [Sales].[SalesOrderHeader]
where year([OrderDate])=2008 and CustomerID>30100 and CustomerID<30118
select*from MsalesOrders
order by customerID
--Tạo trigger
go
create trigger trigger_priority on MsalesOrders
for insert, update, delete
as
```

```
	WITH CTE AS (
		select CustomerId from inserted
		union
		select CustomerId from deleted
	)
	UPDATE Mcustomer
	SET custpriority =
				case
					when t.Total < 10000 then 3
					when t.Total between 10000 and 50000 then 2
					when t.Total > 50000 then 1
					when t.Total IS NULL then NULL
				end
	FROM Mcustomer c INNER JOIN CTE ON CTE.CustomerId = c.CustomerId
				LEFT JOIN (select MsalesOrders.customerID, SUM(SubTotal) Total
					from MsalesOrders inner join CTE
					on CTE.CustomerId = MsalesOrders.CustomerId
					group by MsalesOrders.customerID) t ON t.CustomerId =
c.CustomerId

GO
insert MsalesOrders values(71847, '2016-01-01', 10000, 30112)
select*from Mcustomer
where CustomerId=30112
select*from MsalesOrders

--cau3=============================================================
--Viết một trigger thực hiện trên bảng Memployees sao cho khi người dùng thực hiện
--chèn thêm một nhân viên mới vào bảng Memployees thì chương trình cập nhật số
--nhân viên trong cột NumOfEmployee của bảng MDepartment. Nếu tổng số nhân
--viên của phòng tương ứng <=200 thì cho phép chèn thêm, ngược lại thì hiển thị
--thông báo “Bộ phận đã đủ nhân viên” và hủy giao tác.
--=================================================================
```

```sql
create table MDepartment
(
	DepartmentID int not null primary key,
	Name nvarchar(50),
	NumOfEmployee int
)
insert MDepartment
select [DepartmentID],[Name], null
from [HumanResources].[Department]

create table Memployees
(
	EmployeeID int not null,
	Firtname nvarchar(50),
	MiddleName nvarchar(50),
	LastName nvarchar(50),
	DepartmentID int foreign key references MDepartment(DepartmentID)
	constraint pk_emp_depart primary key(EmployeeID, DepartmentID)
)
insert [Memployees]
select e.[BusinessEntityID], [FirstName],[MiddleName],[LastName], [DepartmentID]
from [HumanResources].[Employee] e join [Person].[Person] p on
e.BusinessEntityID=p.BusinessEntityID
                                        join [HumanResources].[EmployeeDepartmentHistory] h
on e.BusinessEntityID=h.BusinessEntityID
select*from [dbo].[Memployees]
order by [DepartmentID]
go
--tao trigger
create trigger cau3 on [dbo].[Memployees]
for insert
as
	declare @numofEmp int, @DepartID int
```

```
      select @DepartID=i.DepartmentID from inserted i
      set @numofEmp=(select COUNT(*)
                        from [dbo].[Memployees] e
                        where e.DepartmentID=@DepartID
                       )
      if @numofEmp>=180
            begin
                  print 'so nhan vien da du'
                  rollback
            end
      else
            update MDepartment
            set NumOfEmployee =@numofEmp
            where DepartmentID= @DepartID
go
--test
insert [dbo].[Memployees] values(291, 'Nguyen','Hoang','Anh',1)
insert [dbo].[Memployees] values(292, 'Nguyen','Hoang','Thu',2)
--kiem tra ket qua

select *from MDepartment
--cau4================================================================
--Bảng [Purchasing].[Vendor], chứa thông tin của nhà cung cấp, thuộc tính
--CreditRating hiển thị thông tin đánh giá mức tín dụng, có các giá trị:
--1 = Superior
--2 = Excellent
--3 = Above average
--4 = Average
--5 = Below average
--Viết một trigger nhằm đảm bảo khi chèn thêm một record mới vào bảng
--[Purchasing].[PurchaseOrderHeader], nếu Vender có CreditRating=5 thì hiển thị
--thông báo không cho phép chèn và đồng thời hủy giao tác
--====================================================================
```

```sql
select*from [Purchasing].[Vendor]
select*from [Purchasing].[PurchaseOrderHeader]
go
CREATE TRIGGER Purchasing.LowCredit ON Purchasing.PurchaseOrderHeader
AFTER INSERT
AS
IF EXISTS (SELECT *
           FROM Purchasing.PurchaseOrderHeader AS p
                        JOIN inserted AS i ON p.PurchaseOrderID = i.PurchaseOrderID
                        JOIN Purchasing.Vendor AS v ON v.BusinessEntityID = p.VendorID
           WHERE v.CreditRating = 5
          )
BEGIN
     RAISERROR ('A vendor''s credit rating is too low to accept new purchase orders.', 16, 1);
     ROLLBACK TRANSACTION;
END;
GO
--cau 5=============================================================
--Viết một trigger thực hiện trên bảng SalesOrderDetail. Khi chèn thêm một đơn đặt
--hàng vào bảng SalesOrderDetail với số lượng xác định trong field OrderQty, nếu
--số lượng trong kho (ProductInventory
--lưu thông tin số lượng sản phẩm trong
--kho) Quantity> OrderQty thì cập nhật
--lại số lượng trong kho
--Quantity= Quantity - OrderQty,
--ngược lại nếu Quantity = 0 thì xuất
--thông báo “Kho hết hàng” và đồng thời
--hủy giao tác.
--=================================================================
create table MProduct
(
     MProductID int not null primary key,
     ProductName nvarchar(50),
```

```
     ListPrice money
)
insert MProduct (MProductID, ProductName,ListPrice)
select [ProductID], [Name], [ListPrice]
from [Production].[Product]
where [ProductID]<=710
select*from MProduct

create table MSalesOrderHeader
(
     MSalesOrderID int not null primary key,
     OrderDate datetime
)
insert MSalesOrderHeader
select [SalesOrderID], [OrderDate]
from [Sales].[SalesOrderHeader]
where [SalesOrderID] in (select [SalesOrderID] from [Sales].[SalesOrderDetail] where
[ProductID]<=710)
select*from MSalesOrderHeader

create table MSalesOrderDetail
(
     SalesOrderDetailID int IDENTITY(1,1) primary key,
     ProductID int not null foreign key(ProductID) references MProduct(MProductID),
     SalesOrderID int not null foreign key (SalesOrderID) references
MSalesOrderHeader(MSalesOrderID),
     OrderQty int
)
insert MSalesOrderDetail(ProductID, SalesOrderID,OrderQty)
select [ProductID],[SalesOrderID], [OrderQty]
from [Sales].[SalesOrderDetail] where [ProductID] in(select MProductID from MProduct)
--//tạo bảng MProduct_inventory
create table MProduct_inventory
```

```
(
	productID int not null primary key,
	quantity smallint
)

insert MProduct_inventory
select [ProductID],sum([Quantity]) as sumofquatity
	from [Production].[ProductInventory]
	group by [ProductID]
go
----tao trigger
create trigger cau5 on MSalesOrderDetail
for insert
as
	begin
		declare @sldathang int, @sltrongkho int, @masp int
		select @masp =i.ProductID from inserted i
		select @sldathang=i.OrderQty from inserted i
		set @sltrongkho=(select quantity from MProduct_inventory
					where productID=@masp)
		if @sldathang<@sltrongkho
			update MProduct_inventory
			set quantity=quantity-@sldathang
			where ProductID=@masp
		else
			begin
				print 'Het hang'
				rollback tran
			end
	end
select*from MSalesOrderDetail
select*from MSalesOrderHeader
select*from MProduct_inventory
```

```sql
where [ProductID]=708

---thuc thi trigger
delete from [MSalesOrderDetail]
insert [dbo].[MSalesOrderDetail]
values(708, 43661, 300)
--cau6============================================================
--Tạo trigger cập nhật tiền thưởng (Bonus) cho nhân viên bán hàng SalesPerson, khi
--người dùng chèn thêm một record mới trên bảng SalesOrderHeader, theo quy định
--như sau: Nếu tổng tiền bán được của nhân viên có hóa đơn mới nhập vào bảng
--SalesOrderHeader có giá trị >10000000 thì tăng tiền thưởng lên 10% của mức
--thưởng hiện tại
 --================================================================
go
create table M_SalesPerson
(
     SalePSID int not null primary key,
     TerritoryID int,
     BonusPS money
)
create table M_SalesOrderHeader
(
     SalesOrdID int not null primary key,
     OrderDate date,
     SubTotalOrd money,
     SalePSID int foreign key references M_SalesPerson(SalePSID)
)

go
CREATE trigger bonus_emp on [dbo].[M_SalesOrderHeader]
for insert
as
     begin
```

```
            declare @tt float, @spersonID int
            select @spersonID= i.SalePSID  from inserted i
            set @tt=(select sum([SubTotalOrd])
                        from [dbo].[M_SalesOrderHeader]
                        where SalePSID=@spersonID)
            if @tt>10000000
            begin
                  update [dbo].[M_SalesPerson]
                  set BonusPS=BonusPS*1.1
                  where SalePSID=@spersonID
            end
      end
--Test trigger
```

```
--1.Tạo view dbo.vw_Products hiển thị danh sách các sản phẩm từ bảng
--Production.Product và bảng Production.ProductCostHistory. Thông tin bao gồm
--ProductID, Name, Color, Size, Style, StandardCost, EndDate, StartDate
go
create view vw_Products1
as
      select p.ProductID, Name, Color, Size, Style, p.StandardCost, EndDate, StartDate
      from [Production].[Product] p join [Production].[ProductCostHistory] c
      on p.ProductID=c.ProductID

go
select*from [dbo].[vw_Products1]
sp_helptext [vw_Products1]
--2) Tạo view List_Product_view chứa danh sách các sản phẩm có trên 500 đơn đặt
--hàng trong quí 1 năm 2008 và có tổng trị giá >10000, thông tin gồm ProductID,
--Product_name, countofOrderID và Subtotal.
go
create view List_Product_view
as
```

```
      select p.[ProductID], [Name] as Product_name, countofOrderID= count(*),
Subtotal=sum([OrderQty]*[UnitPrice])
      from [Production].[Product] p join  [Sales].[SalesOrderDetail] o on p.ProductID=o.ProductID
                                          join [Sales].[SalesOrderHeader] h on
o.SalesOrderID=h.SalesOrderID
      where datepart(q, [OrderDate])=1 and YEAR([OrderDate])=2008
      group by p.[ProductID], [Name]
      having sum([OrderQty]*[UnitPrice])>10000


--3) Tạo view dbo.vw_CustomerTotals hiển thị tổng tiền bán được (total sales) từ cột
--TotalDue của mỗi khách hàng (customer) theo tháng và theo năm. Thông tin gồm
--CustomerID, YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS
--OrderMonth, SUM(TotalDue).
go
create view vw_CustomerTotals
as
      select CustomerID, YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS OrderMonth,
sumofTotal=SUM(TotalDue)
      from [Sales].[SalesOrderHeader]
      group by CustomerID, YEAR(OrderDate), MONTH(OrderDate)
--4) Tạo view trả về tổng số lượng sản phẩm (total quantity) bán được của mỗi nhân
--viên theo từng năm. Thông tin gồm SalesPersonID, OrderYear, sumOfOrderQty
go
create view view_SumofQty
as
      select SalesPersonID, OrderYear=year([OrderDate]), sumOfOrderQty=sum([OrderQty])
      from [Sales].[SalesOrderHeader] h join [Sales].[SalesOrderDetail] d on
h.SalesOrderID=d.SalesOrderID
      group by SalesPersonID, year([OrderDate])
--5) Tạo view ListCustomer_view chứa danh sách các khách hàng có trên 25 hóa đơn
--đặt hàng từ năm 2007 đến 2008, thông tin gồm mã khách (PersonID) , họ tên
--(FirstName +' '+ LastName as fullname), Số hóa đơn (CountOfOrders).
```

```sql
go
create view ListCustomer_view
as
	select  [CustomerID], FirstName +' '+ LastName as fullname, count(*)
	from [Sales].[SalesOrderHeader] h join [Person].[Person] p on
h.CustomerID=p.BusinessEntityID
	where year([OrderDate])>=2007 AND year([OrderDate])<=2008
	GROUP BY  [CustomerID], FirstName +' '+ LastName
	HAVING count(*)>25

--6) Tạo view ListProduct_view chứa danh sách những sản phẩm có tên bắt đầu với
--‘Bike’ và ‘Sport’ có tổng số lượng bán trong mỗi mỗi năm trên 500 sản phẩm,
--thông tin gồm ProductID, Name, CountofOrderQty, year. (dữ liệu lấy từ các
--bảng Sales.SalesOrderHeader, Sales.SalesOrderDetail, and
--Production.Product)
go
create view ListProduct_view
as
	select p.ProductID, Name, SumofOrderQty=sum([OrderQty]), year([OrderDate]) as yearofOrder
	from [Production].[Product] p join [Sales].[SalesOrderDetail] d on p.ProductID=d.ProductID
                                        join[Sales].[SalesOrderHeader] h on
d.SalesOrderID=h.SalesOrderID
	where Name like 'Bike%' or Name like 'Sport%'
	group by p.ProductID, Name, year([OrderDate])
	having sum([OrderQty])>500
--7) Tạo view List_department_View chứa danh sách các phòng ban có lương (Rate:
--lương theo giờ) trung bình >30, thông tin gồm Mã phòng ban (DepartmentID),
--tên phòng ban (name), Lương trung bình (name). Dữ liệu từ các bảng
--[HumanResources].[Department],
--[HumanResources].[EmployeeDepartmentHistory],
--[HumanResources].[EmployeePayHistory].
go
create view List_department_View
```

```
as
	select d.DepartmentID, name, avgofRate=avg(Rate)
	from [HumanResources].[Department] d join [HumanResources].[EmployeeDepartmentHistory] e on
d.DepartmentID=e.DepartmentID
                                                    join [HumanResources].[EmployeePayHistory]
h on e.BusinessEntityID=h.BusinessEntityID
	group by d.DepartmentID, name
	having avg(Rate)>30
--8) Tạo view Sales.vw_OrderSummary với từ khóa WITH ENCRYPTION gồm
--orderYear (năm của ngày lập), OrderMonth (tháng của ngày lập), OrderTotal
--(tổng tiền). Sau đó xem thông tin và trợ giúp về mã lệnh của view này
go
create view vw_OrderSummary WITH ENCRYPTION
as
	select year([OrderDate]) as orderYear, month([OrderDate]) as OrderMonth,
OrderTotal=sum([OrderQty]*[UnitPrice])
	from [Sales].[SalesOrderHeader] h join [Sales].[SalesOrderDetail] d on
h.SalesOrderID=h.SalesOrderID
	group by year([OrderDate]), month([OrderDate])
go
sp_helptext vw_OrderSummary
--9) Tạo view Production.vwProducts với từ khóa WITH SCHEMABINDING
--gồm ProductID, Name, StartDate,EndDate,ListPrice của bảng Product và bảng
--ProductCostHistory. Xem thông tin của View. Xóa cột ListPrice của bảng
--Product. Có xóa được không? Vì sao?
go
create view vwProducts WITH SCHEMABINDING
as
	select p.ProductID, Name, StartDate,EndDate,ListPrice
	from  [Production].[Product] p join  [Production].[ProductCostHistory] h on
p.ProductID=h.ProductID
---
go
```

```
alter table [Production].[Product]
drop column ListPrice
--10)Tạo view view_Department với từ khóa WITH CHECK OPTION chỉ chứa các
--phòng thuộc nhóm có tên (GroupName) là “Manufacturing” và “Quality
--Assurance”, thông tin gồm: DepartmentID, Name, GroupName.
go
create view view_Department
as
      select DepartmentID, Name, GroupName
      from [HumanResources].[Department]
      where Name='Manufacturing' or Name='Quality Assurance'
      WITH CHECK OPTION
      go
select*from view_Department
--a. Chèn thêm một phòng ban mới thuộc nhóm không thuộc hai nhóm
--“Manufacturing” và “Quality Assurance” thông qua view vừa tạo. Có
--chèn được không? Giải thích
go
insert view_Department values( 'nhan su', 'a')

select *from [HumanResources].[Department]
--b. Chèn thêm một phòng mới thuộc nhóm “Manufacturing” và một phòng
--thuộc nhóm “Quality Assurance”.
insert view_Department values( 'Quality ', 'Quality Assurance')
--c. Dùng câu lệnh select xem kết quả trong bảng Department
----
```

```
SQL Transactions
----------------

Appendix 1 / SQL Server Express

The following scripts are copied from Appendix 1 of the SQL Transactions tutorial
```

so that the examples can be tested easily using copy & paste in a local SQL Server Express, which can be downloaded for free from Microsofts SQL site.
The user should have at least dbcreator server role in the SQL Server instance.
As the client tool we recommend the use of SQL Server Management Studio.

Note: As default SQL Server uses AUTOCOMMIT mode in which we cannot rollback anything. However, by BEGIN TRANSACTION command can be used to start an exlicit transaction.
By SET IMPLICIT_TRANSACTIONS ON the whole SQL session can changed to use implicit transactions.

For the beginning you may want to create a new test database into the instance using the following command

```
CREATE DATABASE TestDB;
```

```
-- Part 1   Experimenting with single transactions
--            - "Logical Units of Work"
```

```
-- Exercise 1.1

USE TestDB;
--  Autocommit mode.  This the default,
--but can be ensured by
SET IMPLICIT_TRANSACTIONS OFF;
--
CREATE TABLE T (id INT NOT NULL PRIMARY KEY, s VARCHAR(30), si SMALLINT);
INSERT INTO T (id, s) VALUES (1, 'first');
ROLLBACK;
SELECT * FROM T;
```

```sql
-- Did ROLLBACK have any effect?

BEGIN TRANSACTION;  -- an explicit transaction begins
INSERT INTO T (id, s) VALUES (2, 'second');
SELECT * FROM T;
ROLLBACK;
SELECT * FROM T;

-- Exercise 1.2
INSERT INTO T (id, s) VALUES (3, 'third');
ROLLBACK;
SELECT * FROM T;
COMMIT;
-- Does SQL Server accept COMMIT or ROLLBACK in AUTOCOMMIT mode?

-- Exercise 1.3
BEGIN TRANSACTION;
DELETE FROM T WHERE id > 1;
COMMIT;
SELECT * FROM T;

-- Exercise 1.4
-- DDL stands for Data Definition Language.  In SQL the statements like
-- CREATE, ALTER and DROP are called DDL statements.
-- Now let's test use of DDL commands in a transaction!
SET IMPLICIT_TRANSACTIONS ON;
INSERT INTO T (id, s) VALUES (2, 'will this be committed?');
CREATE TABLE T2 (id INT);  -- testing use of a DDL command in atransaction!
INSERT INTO T2 (id) VALUES (1);
SELECT * FROM T2;
ROLLBACK;
GO  -- GO marks the end of a batch of SQL commands to be sent to the server
```

```
SELECT * FROM T;  -- What has happened to T ?
SELECT * FROM T2; -- What has happened to T2 ?
-- What we learned from this experiment?

-- Exercise 1.5
DELETE FROM T WHERE id > 1;
COMMIT;
-----------------------------------------------------------------
-- Testing if an error would lead to automatic rollback in SQL Server?
--   @@ERROR is the SQLCode indicator in Transact-SQL, and
--   @@ROWCOUNT is the count indicator of the effected rows
-----------------------------------------------------------------
INSERT INTO T (id, s) VALUES (2, 'The test starts by this');
-- division by zero should fail
SELECT 1/0 AS dummy;   -- division by zero should fail
SELECT @@ERROR AS 'sqlcode'
-- Next updating an non-existing row
UPDATE T SET s = 'foo' WHERE id = 9999;
SELECT @@ROWCOUNT AS 'Updated'
-- and deleting an non-existing row
DELETE FROM T WHERE id = 7777;
SELECT @@ROWCOUNT AS 'Deleted'
COMMIT;
SELECT * FROM T;
--
INSERT INTO T (id, s) VALUES (2, 'Hi, I am a duplicate')
INSERT INTO T (id, s) VALUES (3, 'How about inserting too long string value?')
INSERT INTO T (id, s, si) VALUES (4, 'Smallint overflow for 32769?', 32769);
INSERT INTO T (id, s) VALUES (5, 'Is the transaction still active?');
COMMIT;
-- Did the whole transaction succeed, do we see all inserted rows?
SELECT * FROM T;
GO
```

```
DELETE FROM T WHERE id > 1;
SELECT * FROM T;
COMMIT;

-----------------------------------------------------------------
-- Exercise 1.5b
-- This is special to SQL Server only!
SET XACT_ABORT ON;  -- In this mode an error generates automatic rollback
SET IMPLICIT_TRANSACTIONS ON;
SELECT 1/0 AS dummy;   -- division by zero
INSERT INTO T (id, s) VALUES (6, 'insert after arithm. error');
COMMIT;
GO
SET XACT_ABORT OFF; -- In this mode an error does not generate automatic rollback
SELECT * FROM T;
-- What happened to the transaction?


--=================================================================
-- A1.2   Experimenting with Transaction Logic
-----------------------------------------------------------------
-- Exercise 1.6: COMMIT and ROLLBACK
-----------------------------------------------------------------
SET NOCOUNT ON;  -- skipping the "n row(s) affected" messages
DROP TABLE Accounts;
SET IMPLICIT_TRANSACTIONS ON;
--
CREATE TABLE Accounts (
acctID  INTEGER NOT NULL PRIMARY KEY,
balance INTEGER NOT NULL
        CONSTRAINT unloanable_account CHECK (balance >= 0)
);
COMMIT;
```

```
INSERT INTO Accounts (acctID,balance) VALUES (101,1000);
INSERT INTO Accounts (acctID,balance) VALUES (202,2000);
SELECT * FROM Accounts;
COMMIT;

-- let’s try the bank transfer
UPDATE Accounts SET balance = balance - 100 WHERE acctID = 101;
UPDATE Accounts SET balance = balance + 100 WHERE acctID = 202;
SELECT * FROM Accounts;
ROLLBACK;

-- Let's test the CHECK constraint actually work:
UPDATE Accounts SET balance = balance - 2000 WHERE acctID = 101;
UPDATE Accounts SET balance = balance + 2000 WHERE acctID = 202;
SELECT * FROM Accounts ;
ROLLBACK;

-- Transaction logic
-- using the IF structure of Transact-SQL
SELECT * FROM Accounts;
UPDATE Accounts SET balance = balance - 2000 WHERE acctID = 101;
IF @@error <> 0 OR @@rowcount = 0
   ROLLBACK
ELSE BEGIN
   UPDATE Accounts SET balance = balance + 2000 WHERE acctID = 202;
   IF @@error <> 0 OR @@rowcount = 0
      ROLLBACK
   ELSE
      COMMIT;
   END;

SELECT * FROM Accounts;
COMMIT;
```

```
-- Restoring the original contents
DELETE FROM Accounts;
INSERT INTO Accounts (acctID,balance) VALUES (101,1000);
INSERT INTO Accounts (acctID,balance) VALUES (202,2000);
SELECT * FROM Accounts;
COMMIT;

-- How about using a non-existent bank account
UPDATE Accounts SET balance = balance - 500 WHERE acctID = 101;
UPDATE Accounts SET balance = balance + 500 WHERE acctID = 777;
SELECT * FROM Accounts ;
ROLLBACK;

-- Fixing the case using the IF structure of Transact-SQL
SELECT * FROM Accounts;
UPDATE Accounts SET balance = balance - 500 WHERE acctID = 101;
IF @@error <> 0 OR @@rowcount = 0
   ROLLBACK
ELSE BEGIN
   UPDATE Accounts SET balance = balance + 500 WHERE acctID = 707;
   IF @@error <> 0 OR @@rowcount = 0
      ROLLBACK
   ELSE
      COMMIT;
   END;
SELECT * FROM Accounts;
COMMIT;


-----------------------------------------------------------------
-- Exercise 1.7  Testing the database recovery
-----------------------------------------------------------------
```

```
BEGIN TRANSACTION;
INSERT INTO T (id, s) VALUES (9, 'Just before a soft crash ..');
SELECT * FROM T;

-------------------------------------------------------------
-- On exiting now the Management Studio, we will get question
-- ".. Do you wish to commit these transactions before closing the window?"
-- and for purposes of our experiment we select  ”No”.
-- On restarting Management Studio and connecting to our TestDB we can study
-- what happened to our latest uncommitted transaction just by listing
-- the contents of table T
-------------------------------------------------------------
SET NOCOUNT ON;
SELECT * FROM T;
-- So, can we see the row 9 in the database or has the transaction
-- been rolled back autmatically?

--                        * * *

--=============================================================
-- Part 2   Experimenting with Concurrent Transactions
--=============================================================
-- For concurrency experiments we need to open two parallel
-- SQL query windows having SQL sessions “client A” and “client B”
-- accessing the same database TestDB.  For both sessions we select
-- result to be listed in text mode by following menu selections
--    Query -> Results To -> Results to Text
-- and set both sessions to use implicit transactions by
SET IMPLICIT_TRANSACTIONS ON;

-- For making best visual use of the Management Studio, we can organize
-- the parallel SQLQuery windows appear vertically side-by-side by pressing
-- the alternate mouse button on the title of either SQLQuery window and
```

```
-- selecting from the pop-up window the alternative ”New Vertical Tab Group”
-- (see figure 1-1 Opening 2 SQLQuery windows side-by-side)
----------------------------------------------------------------

-- Exercise 2.1

-- 0. To start with fresh contents we enter following commands on a session
SET IMPLICIT_TRANSACTIONS ON;
DELETE FROM Accounts;
INSERT INTO Accounts (acctID,balance) VALUES (101,1000);
INSERT INTO Accounts (acctID,balance) VALUES (202,2000);
COMMIT;

-- "Simulation of the Lost Update Problem"
--
-- Lost Update problem is raised if some INSERTed or UPDATEd row would be
-- updated or deleted by some concurrent transaction before the first transaction
-- ends. This might be possible in file-based NoSQL solutions, but modern DBMS
-- products will prevent this. However, after the first transaction commits,
-- any competing transaction can overwrite the rows of the committed transaction.
--
-- In the following we simulate the Lost Update scenario using READ COMMITTED
-- isolation, which does not keep the S-locks.  First the client applications
-- read the balance values releasing the S-locks

-- 1. client A starts
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT acctID, balance FROM Accounts WHERE acctID = 101;

-- 2. client B starts
SET NOCOUNT ON;
SET IMPLICIT_TRANSACTIONS ON;
```

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT acctID, balance FROM Accounts WHERE acctID = 101;

-- 3. client A continues
UPDATE Accounts SET balance = 1000 - 200 WHERE acctID = 101;

-- 4. client B continues
UPDATE Accounts SET balance = 1000 - 500 WHERE acctID = 101;

-- 5. without waiting client A continues
SELECT acctID, balance FROM Accounts WHERE acctID = 101;
COMMIT;

-- 6. client B continues
SELECT acctID, balance FROM Accounts WHERE acctID = 101;
COMMIT;

-- Note: The "Blind Overwriting" reliability problem can be solved
--       if UPDATE commands use "sensitive updates", such as
--            SET balance = balance - 500 WHERE ...

-----------------------------------------------------------------
-- Exercise 2.2  "Lost Update Problem" fixed by locks,
--              (competition on a single resource)
-----------------------------------------------------------------
--     Competition on a single resource
--     using SELECT .. UPDATE scenarios both client A and B
--     tries to withdraw amounts from the same account.
--
-- 0. First restoring the original contents by client A
SET IMPLICIT_TRANSACTIONS ON;
DELETE FROM Accounts;
INSERT INTO Accounts (acctID,balance) VALUES (101,1000);
```

```
INSERT INTO Accounts (acctID,balance) VALUES (202,2000);
COMMIT;

-- 1. client A starts
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT acctID, balance FROM Accounts WHERE acctID = 101;

-- 2. client B starts
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT acctID, balance FROM Accounts WHERE acctID = 101;

-- 3. client A continues
UPDATE Accounts SET balance = balance - 200
WHERE acctID = 101;

-- 4. client B continues without waiting for A
UPDATE Accounts SET balance = balance - 500
WHERE acctID = 101;

-- 5. the client which survived will commit
SELECT acctID, balance FROM Accounts WHERE acctID = 101;
COMMIT;

-----------------------------------------------------------------
-- Exercise 2.3   Competition on two resources in different order
--                using UPDATE-UPDATE scenarios
-----------------------------------------------------------------
--
-- Client A transfers 100 euros from account 101 to 202
-- Client B transfers 200 euros from account 202 to 101
--
```

```sql
-- 0. First restoring the original contents by client A
SET IMPLICIT_TRANSACTIONS ON;
DELETE FROM Accounts;
INSERT INTO Accounts (acctID,balance) VALUES (101,1000);
INSERT INTO Accounts (acctID,balance) VALUES (202,2000);
COMMIT;

-- 1. client A starts
UPDATE Accounts SET balance = balance - 100
WHERE acctID = 101;

-- 2. Client B starts
SET IMPLICIT_TRANSACTIONS ON;
UPDATE Accounts SET balance = balance - 200
WHERE acctID = 202;

-- 3. Client A continues
UPDATE Accounts SET balance = balance + 100
WHERE acctID = 202;

-- 4. Client B continues
UPDATE Accounts SET balance = balance + 200
WHERE acctID = 101;

-- 5. Client A continues if it can ..
COMMIT;

-- 6. Client B continues if it can ..
COMMIT;

-----------------------------------------------------------------
-- In the following we will experiment with concurrency anomalies i.e.
-- data reliability risks known by ISO SQL standard.
```

```
-- First play with the experiment, see the results, and then
--  TRY TO FIX  the experiment to avoid the anomaly

-----------------------------------------------------------------
-- Exercise 2.4     Dirty Read ?
-----------------------------------------------------------------

-- 0. First restoring the original contents by client A
SET IMPLICIT_TRANSACTIONS ON;
DELETE FROM Accounts;
INSERT INTO Accounts (acctID,balance) VALUES (101,1000);
INSERT INTO Accounts (acctID,balance) VALUES (202,2000);
COMMIT;

-- 1. client A starts
SET IMPLICIT_TRANSACTIONS ON;
UPDATE Accounts SET balance = balance - 100 WHERE acctID = 101;
UPDATE Accounts SET balance = balance + 100 WHERE acctID = 202;

-- 2. Client B starts
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM Accounts;
COMMIT;

-- 3. Client A continues
ROLLBACK;
SELECT * FROM Accounts;
COMMIT;

-----------------------------------------------------------------
-- Exercise 2.5     Non-repeatable Read ?
-----------------------------------------------------------------
```

```
-- In non-repeatable read anomaly some rows read in the current transaction
-- may not appear in the resultset if the read operation would be repeated
-- before end of the transaction.

-- 0. First restoring the original contents by client A
SET IMPLICIT_TRANSACTIONS ON;
DELETE FROM Accounts;
INSERT INTO Accounts (acctID,balance) VALUES (101,1000);
INSERT INTO Accounts (acctID,balance) VALUES (202,2000);
COMMIT;

-- 1. client A starts
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
--  Listing accounts having balance > 500 euros:
SELECT * FROM Accounts WHERE balance > 500;

-- 2. Client B starts
SET IMPLICIT_TRANSACTIONS ON;
UPDATE Accounts SET balance = balance - 500 WHERE acctID = 101;
UPDATE Accounts SET balance = balance + 500 WHERE acctID = 202;
COMMIT;

-- 3. Client A continues
-- Can we see the same accounts and balances as in step 1?
SELECT * FROM Accounts WHERE balance > 500;
COMMIT;

-----------------------------------------------------------------
-- Exercise 2.6     Insert Phantom ?
-----------------------------------------------------------------
-- Insert phantoms are rows inserted by concurrent transactions and
-- which the current might see before the end of the transaction.
```

```sql
--
-- 0. First restoring the original contents by client A
SET IMPLICIT_TRANSACTIONS ON;
DELETE FROM Accounts;
INSERT INTO Accounts (acctID,balance) VALUES (101,1000);
INSERT INTO Accounts (acctID,balance) VALUES (202,2000);
COMMIT;

-- 1. client A starts
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
-- Accounts having balance > 1000 euros:
SELECT * FROM Accounts WHERE balance > 1000;

-- 2. Client B starts
SET IMPLICIT_TRANSACTIONS ON;
INSERT INTO Accounts (acctID,balance) VALUES (303,3000);
COMMIT;

-- 3. Client A continues
-- Let’s see the results:
SELECT * FROM Accounts WHERE balance > 1000;

COMMIT;

-- Task: How can we prevent the Phantoms?

--=============================================================
-- Snapshot studies
-------------------------------------------------------------
-- The database need to be configured to support SNAPSHOT isolation.
-- For this we create a new database
--
CREATE DATABASE SnapsDB;  -- to be configured to support snapshots
```

```
-- The database options  READ_COMMITTED_SNAPSHOT and
-- ALLOW_SNAPSHOT_ISOLATION need to be set ON !
--
-- Then both client A and B are switched to use SnapsDB
USE SnapsDB;
-----------------------------------------------------------------
-- Exercise 2.7    A Snapshot study with different kinds of Phantoms
-----------------------------------------------------------------
-- 0. Setup the test
DROP TABLE T;
GO
SET IMPLICIT_TRANSACTIONS ON;
SET NOCOUNT ON;
CREATE TABLE T (id INT NOT NULL PRIMARY KEY, s VARCHAR(30), i SMALLINT);
COMMIT;
--
DELETE FROM T;
INSERT INTO T (id, s, i) VALUES (1, 'first',  1);
INSERT INTO T (id, s, i) VALUES (2, 'second', 2);
INSERT INTO T (id, s, i) VALUES (3, 'third', 1);
INSERT INTO T (id, s, i) VALUES (4, 'forth', 2);
INSERT INTO T (id, s, i) VALUES (5, 'to be or not to be', 1);
COMMIT;

-- 1. client A starts
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION LEVEL SNAPSHOT ;
SELECT * FROM T WHERE i = 1;

-- 2. Client B starts,
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

```sql
INSERT INTO T (id, s, i) VALUES (6, 'Insert Phantom', 1);
UPDATE T SET s = 'Update Phantom', i = 1 WHERE id = 2;
DELETE FROM T WHERE id = 5;
SELECT * FROM T;

-- 3. Client A continues
-- Let’s repeat the query and try some updates
SELECT * FROM T WHERE i = 1;
INSERT INTO T (id, s, i) VALUES (7, 'inserted by A', 1);
UPDATE T SET s = 'update by A inside snapshot' WHERE id = 3;
UPDATE T SET s = 'update by A outside snapshot' WHERE id = 4;
UPDATE T SET s = 'update by A after B' WHERE id = 1;
SELECT * FROM T WHERE i = 1;

-- 3.5 . Client C queries
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM T;


-- 4. Client B continues
SELECT * FROM T;

-- 5. Client A continues
SELECT * FROM T WHERE i = 1;
UPDATE T SET s = 'update after delete?' WHERE id = 5;

-- 6. Client B continues without waiting for A
COMMIT;
SELECT * FROM T;

-- 7. Client A continues if it can
SELECT * FROM T WHERE i = 1;
COMMIT;
```

```sql
-- 8. Client B reads the final state
SELECT * FROM T;

-- Task:  Explain how the experiment proceeded

--          ** End of exercises **

--=============================================================
-- Experimenting with the BankTransfer as a stored procedure
-------------------------------------------------------------
USE TestDB;
DROP PROCEDURE BankTransfer;
GO

SET IMPLICIT_TRANSACTIONS ON;
CREATE PROCEDURE BankTransfer
   (@fromAcct INT,
    @toAcct   INT,
    @amount   INT,
    @msg     VARCHAR(50) OUTPUT)
AS
BEGIN
    SET @msg = 'OK';
    UPDATE Accounts SET balance = balance - @amount WHERE acctID = @fromAcct;
    IF @@ERROR <> 0 OR @@ROWCOUNT = 0 BEGIN
        ROLLBACK;
        SET @msg = '* Unknown from account ' + CAST(@fromAcct AS VARCHAR(10));
    END
    ELSE BEGIN
        UPDATE Accounts SET balance = balance + @amount WHERE acctID = @toAcct;
        IF @@ERROR <> 0 OR @@ROWCOUNT = 0 BEGIN
            ROLLBACK;
```

```
            SET @msg = '* Unknown from account ' + CAST(@toAcct  AS VARCHAR(10));
        END;
    END;
END;
-----------------------------------------------------------------
Testing the procedure:
-----------------------------------------------------------------
-- First restoring the original contents by client A
SET IMPLICIT_TRANSACTIONS OFF;
DELETE FROM Accounts;
INSERT INTO Accounts (acctID,balance) VALUES (101,1000);
INSERT INTO Accounts (acctID,balance) VALUES (202,2000);

BEGIN TRANSACTION;
BEGIN
  DECLARE @out VARCHAR(50);
  EXEC BankTransfer @fromAcct=101,@toAcct=202,@amount=100,@msg=@out OUTPUT;
  IF @out='OK' COMMIT;
  SELECT 'msg='+@out AS outmsg;
  SELECT * FROM Accounts;
END;

(1 row(s) affected)

(1 row(s) affected)
outmsg
-----------------------------------------------------
msg=OK

(1 row(s) affected)

acctId      balance
----------- ---------------------------------------
```

```
101         900.00
202         2100.00

(2 row(s) affected)


BEGIN TRANSACTION;
BEGIN
  DECLARE @out VARCHAR(50);
  EXEC BankTransfer @fromAcct=100,@toAcct=202,@amount=100,@msg=@out OUTPUT;
  IF @out='OK' COMMIT;
  SELECT 'msg='+@out AS outmsg;
  SELECT * FROM Accounts;
END;

(0 row(s) affected)
Msg 266, Level 16, State 2, Procedure BankTransfer, Line 0
Transaction count after EXECUTE indicates a mismatching number of BEGIN and COMMIT statements.
Previous count = 1, current count = 0.
outmsg
----------------------------------------------------
msg=* Unknown from account 100

(1 row(s) affected)

acctId      balance
----------- ---------------------------------------
101         900.00
202         2100.00

(2 row(s) affected)
```

```
BEGIN TRANSACTION;
BEGIN
  DECLARE @out VARCHAR(50);
  EXEC BankTransfer @fromAcct=101,@toAcct=200,@amount=100,@msg=@out OUTPUT;
  IF @out='OK' COMMIT;
  SELECT 'msg='+@out AS outmsg;
  SELECT * FROM Accounts;
END;

(1 row(s) affected)

(0 row(s) affected)
Msg 266, Level 16, State 2, Procedure BankTransfer, Line 0
Transaction count after EXECUTE indicates a mismatching number of BEGIN and COMMIT statements.
Previous count = 1, current count = 0.
outmsg
-----------------------------------------------------
msg=* Unknown from account 200

(1 row(s) affected)

acctId      balance
----------- ---------------------------------------
101         900.00
202         2100.00

(2 row(s) affected)

BEGIN TRANSACTION;
BEGIN
  DECLARE @out VARCHAR(50);
  EXEC BankTransfer @fromAcct=101,@toAcct=202,@amount=2000,@msg=@out OUTPUT;
```

```
  IF @out='OK' COMMIT;
  SELECT 'msg='+@out AS outmsg;
  SELECT * FROM Accounts;
END;

Msg 547, Level 16, State 0, Procedure BankTransfer, Line 9
The UPDATE statement conflicted with the CHECK constraint "CK__Accounts__balanc__628FA481". The
conflict occurred in database "TestDB", table "dbo.Accounts", column 'balance'.
The statement has been terminated.
Msg 266, Level 16, State 2, Procedure BankTransfer, Line 0
Transaction count after EXECUTE indicates a mismatching number of BEGIN and COMMIT statements.
Previous count = 1, current count = 0.
outmsg
-----------------------------------------------------
msg=* Unknown from account 101

(1 row(s) affected)

acctId      balance
----------- --------------------------------------
101         900.00
202         2100.00

(2 row(s) affected)

------------------------------ end of SQL Server experiment -----------------
```