use sample;

go

--Example showing implementation of transaction

-- both update statements are committed or rolled back together as a single unit

begin transaction

update employee

SET dept\_no ='D1'

WHERE emp\_no = 15000;

update works\_on

SET project\_no ='p3'

Where emp\_no =15000

and project\_no ='p1'

if @@ERROR <> 0

begin

print 'Aborted, there is an error'

ROLLBACK

end

else

commit;

/\*The SAVE TRANSACTION statement sets a save point within a

transaction. A save point marks a specified point within the transaction so

that all updates that follow can be canceled without canceling the entire

transaction. (To cancel an entire transaction, use the ROLLBACK

statement.) \*/

BEGIN TRANSACTION

insert into department ( dept\_no, dept\_name)

values ( 'D6', 'Support')

SAVE TRANSACTION a

insert into department ( dept\_no, dept\_name)

values ( 'D7', 'Research')

SAVE TRANSACTION b

insert into department ( dept\_no, dept\_name)

values ( 'D8', 'Management')

ROLLBACK TRANSACTION b

insert into department ( dept\_no, dept\_name)

values ( 'D8', 'Sales')

ROLLBACK TRANSACTION a

COMMIT TRANSACTION;

/\* Only the first insert is committed. All others are cancelled

DBCC USEROPTIONS -- to check isolation levels. SQL Server defaults to "read committed"

/\*Updating a row puts an exclusive row level locking on the table\*/

begin tran

update employee

set emp\_fname ='Matt'

where emp\_no =28559

-- rollback

sp\_lock -- to see the locking resources.

/\* Reading a table with exclusive lock, it must wait until the lock is released\*/

select \*

from employee

where dept\_no ='D1'

/\* Reading a table with exclusive lock, it has to wait until the lock is released

NOLOCK hint tells the Engine it is ok to read rows that are not yet committed\*/

select \*

from employee (nolock)

where dept\_no ='D1'

/\* IF you don’t want to wait, set the timeout to wait in milliseconds\*/

SET LOCK\_TIMEOUT 800

select \*

from employee

where dept\_no ='D1'