## Murach Chapter 16 How to Work With Transactions

**1: Use the db2 database**

Create a transaction that will insert a row with the order header information into the Sales.Orders table and a few rows with the order lines information into the Sales.OrderDetails table. The new order ID is produced automatically by SQL Server because the orderid column has an identity property. Immediately after the code inserts the new row into the Sales.Orders table, it stores the newly generated order ID in a local variable, and then it uses that local variable when inserting rows into the Sales.OrderDetails table. Use the following values for the new row in Sales.Orders table:

85,5,getdate(),getdate(),getdate(),3,32.38,'Ship to 85-B','6789 rue de l''Abbaye', 'Reims', null, '10345', 'France'

Use the following values for the 3 new rows in Sales.OrderDetails table:

@neworderid, 11, 14.00, 12, 0.000

@neworderid, 42, 9.80, 10, 0.000

@neworderid, 72, 34.80, 5, 0.000

For test purposes, add a SELECT statement that returns the order ID of the newly generated order. Use the following SELECT statements to test your code:

select \* from (Select top 3 \* From [Sales].[Orders] Order By orderid Desc) A Order By orderid

select \* from (Select top 3 \* From [Sales].[OrderDetails] Order By orderid Desc) A Order By ordered

Answer:

BEGIN TRAN Yoon

INSERT INTO sales.Orders values (85,5,getdate(),getdate(),getdate(),3,32.38,'Ship to 85-B','6789 rue de l''Abbaye', 'Reims', null, '10345', 'France')

DECLARE @neworderid as INT

SET @neworderid = 11078

insert into Sales.OrderDetails VALUEs(@neworderid, 11, 14.00, 12, 0.000)

insert into Sales.OrderDetails VALUEs(@neworderid, 42, 9.80, 10, 0.000)

insert into Sales.OrderDetails VALUEs(@neworderid, 72, 34.80, 5, 0.000)

SELECT orderid from sales.Orders where orderid = @neworderid

commit TRAN Yoon

Text

Description automatically generated

PART 2

Table

Description automatically generated

PART3

***Table

Description automatically generated***

PART4

Graphical user interface, application

Description automatically generated

**2: Use the db2 database**

Create an explicit transactions within your code that will commit the data changes only upon meeting certain criteria; otherwise, the data changes should not occur.

You can use explicit transactions to COMMIT or ROLLBACK a data modification depending on the return of an error in a batch of statements.

Create a local variable to hold the value of the @@ERROR function (which captures the latest error state of a SQL statement).

Use 2 insert statements,

the first statement attempted an INSERT into the HR.Departments table.

There was a unique key on the department\_id, but because the department\_id didn’t already exist in the table, the insert succeeded.

INSERT INTO HR.departments (department\_id, department\_name, Manager\_id, Location\_id) VALUES (280,'Accounts Payable', 200, 1700);

In between these two insert commands you should catch your error.

Another insert is attempted, this time for a department that already exists in the table. Because the table has a primary key column, department\_id this insert will fail:

INSERT INTO HR.departments (department\_id, department\_name, Manager\_id, Location\_id) VALUES (270,'Payroll', null, 1700);

The following statement in this example validated the count of rows in the HR.Departments table, returning 27 rows.

-- Before count: SELECT BeforeCount = COUNT(\*) FROM HR.Departments;  
(See example file for results.)

select \* from (Select top 10 \* From hr.departments Order By department\_id Desc) A Order By department\_id; (See example file for results.)

Another count is performed after the rollback, and again, there are only 27 rows in the database. This is because both INSERTS were in the same transaction and one of the INSERTS failed. Since a transaction is all-or-nothing, no rows were inserted.

-- After count: SELECT AfterCount = COUNT(\*) FROM HR.Departments;  
(See example file for results.)

select \* from (Select top 10 \* From hr.departments Order By department\_id Desc) A Order By department\_id; (See example file for results.)

Answer:

DECLARE @catchError VARCHAR(75)

BEGIN TRY

BEGIN TRAN checkentry

INSERT INTO hr.departments(department\_id, department\_name, manager\_id, location\_id)

VALUES(280,'Accounts Payable', 200, 1700)

INSERT INTO HR.departments (department\_id, department\_name, Manager\_id, Location\_id)

VALUES (270,'Payroll', null, 1700);

SET @catchError = @@ERROR

COMMIT TRAN checkentry

END TRY

BEGIN CATCH

ROLLBACK TRAN checkentry;

THROW

END CATCH

Graphical user interface, text, application, email

Description automatically generated

Table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Table

Description automatically generated

**3: Use the db2 database & the test database**

Create a stored procedure that uses a transaction to maintain data consistency. Use the two related tables, HR.Departments and HR.Employees, with the stored procedure deleting a specific department.

Each Employees record has a foreign key constraint to the Departments table. Therefore, in order to delete a record from the Departments table all associated records from the Employees table must be deleted first. We want to ensure that either all of the associated employees and the department are deleted, or none of the records are deleted, wrap these two DELETE statements within a transaction. Use the db2 database first as show below:

use db2;

go

exec dbo.DeleteDepartment 90; (See example file for results.)

select \* from hr.departments where department\_id = 90; (See example file for results.)

select \* from hr.employees where department\_id = 90; (See example file for results.)

Then run the attached script "Script for 3 in assignment.sql" it will create a new database called test. Rename you procedure to dbo.DeleteDepartmentTest, and execute the procedure using the test database. The test database just contains the tables Departments and Employees. It removes all other constraints that were giving you errors when executing it on the db2 database.

use test;

go

exec dbo.DeleteDepartmentTest 90; (See example file for results.)

select \* from hr.departments where department\_id = 90; (See example file for results.)

select \* from hr.employees where department\_id = 90; (See example file for results.)

Answer:

Graphical user interface, application

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Graphical user interface, application

Description automatically generated

