Logo, company name

Description automatically generated

**School of Computer Science**

**Masters in Applied Computing (M.A.C)**

**Subject Code: COMP8157**

**Subject Name: Advanced Database Topics**

**Professor Dr. Shafaq Khan**

**Lab 3**

**by**

**Hamza Baig (110089314)**

Table of Contents

[In-class Part: 3](#_Toc129294081)

[Create a transaction: 3](#_Toc129294082)

[Let’s run the stored procedure that we created in the previous section. 3](#_Toc129294083)

[Start the transaction at the beginning and end the transaction at the end 3](#_Toc129294084)

[Transaction save point: 6](#_Toc129294085)

[i. Open Transactions: 6](#_Toc129294086)

[ii. Begin transaction: 6](#_Toc129294087)

[iii. Open transactions: 6](#_Toc129294088)

[iv. Nested Transaction: 7](#_Toc129294089)

[v. Save point: 7](#_Toc129294090)

[Automatically roll back transaction: 8](#_Toc129294091)

[Lab 3 Quiz: 11](#_Toc129294092)

[Create database: 11](#_Toc129294093)

[Create Tables: 11](#_Toc129294094)

[Testing: 14](#_Toc129294095)

# In-class Part:

## Create a transaction:

### Let’s run the stored procedure that we created in the previous section.

CREATE OR ALTER PROCEDURE Warehouse.uspInsertColor (@Color AS nvarchar(100))

AS

DECLARE @ColorID INT

SET @ColorID = (SELECT MAX(ColorID) FROM Warehouse.Colors)+1;

INSERT INTO Warehouse.Colors (ColorID, ColorName, LastEditedBy) VALUES (@ColorID, @Color, 1);

SELECT \* FROM Warehouse.Colors WHERE ColorID = @ColorID ORDER BY ColorID DESC;

**Output:**

Graphical user interface, text, application, email

Description automatically generated

### Start the transaction at the beginning and end the transaction at the end

BEGIN TRANSACTION FirstTransaction WITH MARK;

**Output:**

**Graphical user interface, text, application, email

Description automatically generated**

It is important to name a transaction so that it makes it easier to rollback or undo a series of statements under that transaction.

EXEC Warehouse.uspInsertColor 'Sunset Orange';  EXEC Warehouse.uspInsertColor 'Tomato Red';

SELECT \* FROM Warehouse.Colors  ORDER BY ColorID DESC;

Output:

Graphical user interface, text, application, email

Description automatically generated

Running above query in a new window produces no results in no results in my case.

Graphical user interface, text, application, email

Description automatically generated

The Rollback was called in another window, which produced the results in the previous window.

ROLLBACK TRANSACTION FirstTransaction;

Graphical user interface, text, application, email

Description automatically generated

Result in the other window:

Table

Description automatically generated

Committing the transaction does not cause the query in the other window to finish. It completes its execution.

BEGIN TRANSACTION FirstTransaction WITH MARK;

EXEC Warehouse.uspInsertColor 'Sunset Orange';

EXEC Warehouse.uspInsertColor 'Tomato Red';

SELECT \* FROM Warehouse.Colors  ORDER BY ColorID DESC;

COMMIT;

Graphical user interface, text, application, email

Description automatically generated

Table

Description automatically generated

## Transaction save point:

### Open Transactions:

SELECT @@TRANCOUNT AS 'Open Transactions';

Graphical user interface, application

Description automatically generated

### Begin transaction:

BEGIN TRANSACTION;

### Open transactions:

SELECT @@TRANCOUNT AS 'Open Transactions';

Graphical user interface, application

Description automatically generated

### Nested Transaction:

-- BEGIN TRANSACTION;

SELECT @@TRANCOUNT AS 'Open Transactions';

Graphical user interface, text, application

Description automatically generated

### Save point:

EXEC Warehouse.uspInsertColor 'Lemongrass Green';

Graphical user interface, text, application

Description automatically generated

SAVE TRANSACTION SavePointOne;

Graphical user interface, text, application, email

Description automatically generated

SELECT @@TRANCOUNT AS 'Open Transactions';

Graphical user interface, application

Description automatically generated

The count is still 2, as the transaction are nested. We are still under the second transaction.

EXEC Warehouse.uspInsertColor 'Galaxy Purple';

Graphical user interface, text, application, email

Description automatically generated

The added information is shown above.

ROLLBACK TRANSACTION SavePointOne;

Graphical user interface, text, application, email

Description automatically generated

It still shows 2 transactions:

Graphical user interface, application

Description automatically generated

-- COMMIT;

SELECT \* FROM Warehouse.Colors ORDER BY ColorID DESC;

Table

Description automatically generated with low confidence

After the Commit, the colors were instantly shown after the next query was run.

### Automatically roll back transaction:

When a command generates an error in a transaction, the transaction will throw error at that command but since, using transactions, we can use exception handling.

The most recent color added is Lemongrass Green, Galaxy Purple was not added.

-- BEGIN TRANSACTION newone;

EXEC Warehouse.uspInsertColor 'burnished bronze';

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Running the command again:

Graphical user interface, text, application

Description automatically generated

The error is thrown because of unique key violation.

COMMIT Transaction newone;

Graphical user interface, text, application

Description automatically generated

SELECT \* FROM Warehouse.Colors ORDER BY ColorID DESC;

Table

Description automatically generated with low confidence

Graphical user interface, text, application, email

Description automatically generated

BEGIN transaction secondone;

EXEC Warehouse.uspInsertColor 'Glittering Gold';

EXEC Warehouse.uspInsertColor 'Glittering Gold';

COMMIT TRANSACTION secondone;

Graphical user interface, text, application, email

Description automatically generated

No glittering gold is not added:

Table

Description automatically generated with medium confidence

# Lab 3 Quiz:

### Create database:

CREATE DATABASE hamzaADTLab3;

Graphical user interface, text, application, email

Description automatically generated

### Create Tables:

CREATE TABLE hamzaOrders (

    order\_id INT IDENTITY(1,1) PRIMARY KEY,

    order\_date DATE,

    product\_id INT,

    quantity INT,

    total DECIMAL(10,2)

);

-- Create a table for inventory

CREATE TABLE hamzaInventory (

  Productid INT IDENTITY(1,1) PRIMARY KEY,

  product\_name VARCHAR(50),

  stock\_count INT,

  price DECIMAL(10, 2)

);

--Insert sample data into inventory table

INSERT INTO hamzaInventory (product\_name, stock\_count, price) VALUES ('Shirt', 50, 19.99);

Graphical user interface, text, application, email

Description automatically generated

INSERT INTO hamzaInventory (product\_name, stock\_count, price) VALUES ('Pants', 30, 29.99);

INSERT INTO hamzaInventory (product\_name, stock\_count, price) VALUES ('Socks', 100, 4.99);

Graphical user interface, text, application, email

Description automatically generated

SELECT \* FROM hamzaInventory;

Table

Description automatically generated

CREATE PROCEDURE hamza\_spPlaceOrder

@productid INT,

@quantity INT

AS

BEGIN

SET NOCOUNT ON;

SET XACT\_ABORT ON;

BEGIN TRY

BEGIN TRANSACTION;

-- Insert record into the orders table

DECLARE @order\_id INT;

INSERT INTO hamzaOrders(order\_date, product\_id, quantity)

VALUES (GETDATE(), @productid, @quantity);

-- storing orderid to newly added record

SET @order\_id = SCOPE\_IDENTITY();

-- get existing stock count

DECLARE @stockCount INT, @price DECIMAL(10,2);

SELECT @stockCount = stock\_count, @price = price

FROM hamzaInventory WHERE Productid = @productid;

IF @stockCount >= @quantity

BEGIN

-- update the inventory

UPDATE hamzaInventory

SET stock\_count = stock\_count - @quantity

WHERE Productid = @productid;

UPDATE hamzaOrders

SET total = @quantity \* @price

WHERE order\_id = @order\_id;

COMMIT TRANSACTION;

PRINT 'Order Placed Successfully.';

END

ELSE

BEGIN

ROLLBACK TRANSACTION;

PRINT 'Insufficient stock to fulfil the order.';

END

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

BEGIN

ROLLBACK TRANSACTION;

END

PRINT ERROR\_MESSAGE();

END CATCH

END

GO

Text

Description automatically generated

### Testing:

exec hamza\_spPlaceOrder 1,5;

select \* from hamzaOrders;

select \* from hamzaInventory

Table

Description automatically generated

exec hamza\_spPlaceOrder 3,10;

select \* from hamzaOrders;

select \* from hamzaInventory

Table

Description automatically generated

exec hamza\_spPlaceOrder 2,40;

Graphical user interface, text, application, email

Description automatically generated

select \* from hamzaOrders;

select \* from hamzaInventory;

Table

Description automatically generated