

## LAB 6 - Routines

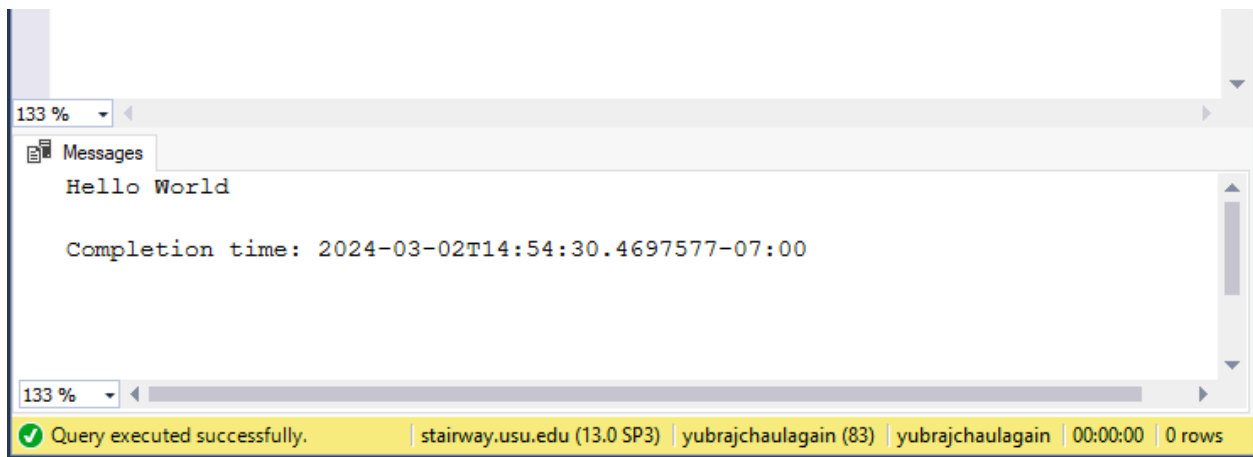
Use the Antiques dataset for the following.

1. Question removed. Please move to next question :)
2. Declare a variable named HelloWorld as a character data type and assign it immediately to 'Hello World.' Next, Print the variable to display 'Hello World'.

```
DECLARE @HelloWorld VARCHAR(255)
```

```
SET @HelloWorld = 'Hello World'
```

```
PRINT @HelloWorld
```



3. Examine the following two coding examples. Are they equivalent? What will happen when you run each one (i.e., will one fail)?

```
Alter TABLE Customer ADD Nickname char(100);
```

```
GO
```

```
SELECT LastName, Nickname FROM Customer WHERE STATE='OR';
```

```
Alter TABLE Customer ADD Nickname char(100);
```

```
SELECT LastName, Nickname FROM Customer WHERE STATE='OR';
```

### **Answer:**

The two coding examples provided are not equivalent due to the presence of the "GO" keyword in the first example.

In SQL Server Management Studio (SSMS) or similar tools, the "GO" keyword is used as a batch separator. It signifies the end of one batch of SQL commands and the beginning of another. When you execute the first example

The "ALTER TABLE" statement will execute successfully, adding the "Nickname" column to the "Customer" table. Then, the second batch containing the "SELECT" statement will execute, attempting to retrieve "LastName" and "Nickname" from the "Customer" table where the state is 'OR'. However, since the "Nickname" column has just been added, it may not contain any data, so the query will still execute, but it may return NULL values for the "Nickname" column.

In the second, There is no "GO" keyword separating the "ALTER TABLE" and "SELECT" statements. This means both statements will be treated as part of the same batch. Therefore, if the "Customer" table exists and does not already have a "Nickname" column, the "ALTER TABLE" statement will execute successfully, adding the "Nickname" column. Then, the "SELECT" statement will execute immediately afterward, attempting to retrieve "LastName" and "Nickname" from the "Customer" table where the state is 'OR'. This may fail if the "Nickname" column has not been added yet, resulting in an error.

So, while both examples aim to achieve the same goal of adding a "Nickname" column to the "Customer" table and then querying data from it, they are not functionally equivalent due to the presence or absence of the "GO" keyword, which affects the execution as separate batches.

4. Write a user-defined function that takes as its input (@InputDate), a date and returns a date 30 days from the input date.

```
CREATE FUNCTION Add30Days (@InputDate DATE) RETURNS DATE AS
```

```
BEGIN
```

```
    DECLARE @OutputDate DATE;
```

```
    SET @OutputDate = DATEADD(DAY, 30, @InputDate);
```

```
    RETURN @OutputDate;
```

```
END;
```

```
DECLARE @InputDate DATE = '2024-03-05';
```

```
SELECT dbo.Add30Days(@InputDate) AS OutputDate;
```

```
CREATE FUNCTION Add30Days (@InputDate DATE) RETURNS DATE AS
BEGIN
    DECLARE @OutputDate DATE;
    SET @OutputDate = DATEADD(DAY, 30, @InputDate);
    RETURN @OutputDate;
END;

DECLARE @InputDate DATE = '2024-03-05';
SELECT dbo.Add30Days(@InputDate) AS OutputDate;
```

OutputDate
2024-04-04

Query executed successfully. | stairway.usu.edu (13.0 SP3) | yubrajchaulagain (55) | yubrajchaulagain | 0

5. Write a stored procedure that takes EmployeeID and TrainingID as arguments. Create the procedure to return the grade of the employee. Find the Grade of Employee 1 for Training number 2.

CREATE PROCEDURE GetEmployeeTrainingGrade

(

@EmployeeID INT,

@TrainingID INT

)

AS

SELECT Grade

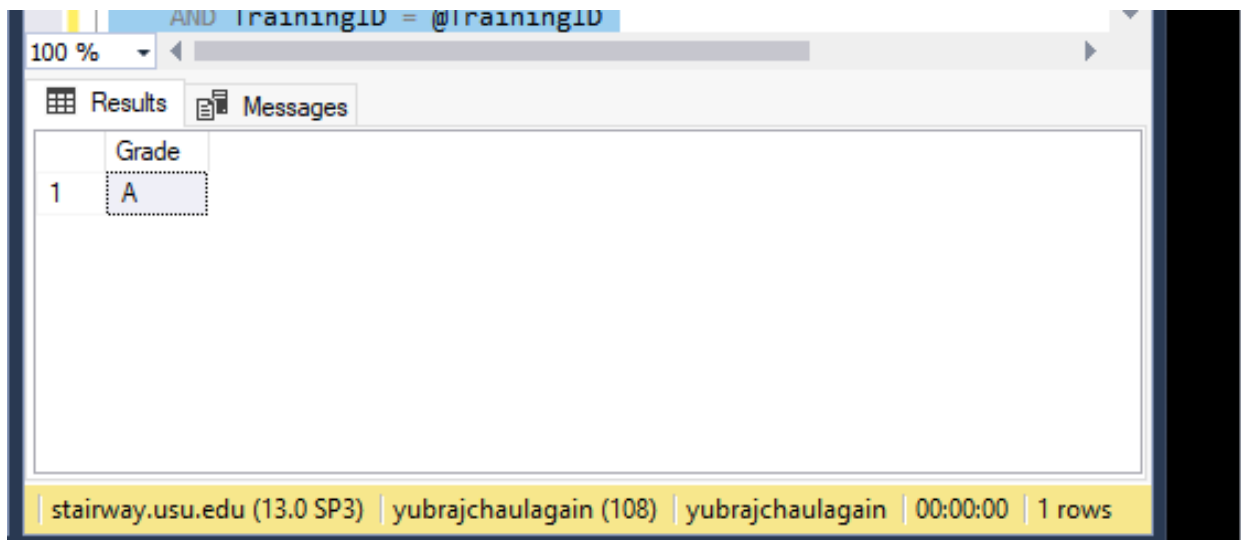
```
FROM EMPLOYEE_TRAINING
```

```
WHERE EmployeeID = @EmployeeID
```

```
AND TrainingID = @TrainingID
```

```
GO
```

```
EXEC GetEmployeeTrainingGrade @EmployeeID = 1, @TrainingID = 2;
```



	Grade
1	A

stairway.usu.edu (13.0 SP3) | yubrajchaulagain (108) | yubrajchaulagain | 00:00:00 | 1 rows

6. Create a Trigger for when an Employee is added to the Employee Table. Use the following DDL to help create the table.

```
CREATE TABLE Employee_Audit
```

```
(
```

```
CreationTime DATETIME NOT NULL DEFAULT(SYSDATETIME()),
```

```
FirstName CHAR(100) NOT NULL,
```

City CHAR(100) NOT NULL

);

GO

Use the following code to test the trigger.

INSERT INTO Employee (FirstName, LastName, State, City, Gender, Salary)

VALUES ('Millie', 'Mills', 'UT', 'Logan', 'Female', 2000000);

**ANSWER:**

CREATE TABLE Employee\_Audit

(

CreationTime DATETIME NOT NULL DEFAULT(SYSDATETIME()),

FirstName CHAR(100) NOT NULL,

City CHAR(100) NOT NULL

);

GO

CREATE TRIGGER trgEmployeeAdded

ON Employee

AFTER INSERT

AS

SET NOCOUNT ON;

INSERT INTO Employee\_Audit (CreationTime, FirstName, City)

SELECT SYSDATETIME(), FirstName, City

FROM inserted;

GO

-- Assuming the Employee table exists with the specified columns

INSERT INTO Employee (FirstName, LastName, State, City, Gender, Salary)

VALUES ('Millie', 'Mills', 'UT', 'Logan', 'Female', 2000000);

SELECT \* FROM Employee\_Audit;

-- Assuming the Employee table exists with the specified columns				
10 %				
Results	Messages			
	CreationTime	FirstName	City	
1	2024-03-06 15:01:31.053	Millie	Logan	
stairway.usu.edu (13.0 SP3)   yubrajchaulagain (108)   yubrajchaulagain   00:00:00   1 rows				

7. Modify the code from the previous question in order to CREATE a trigger whenever an Employee is deleted. Continue to use the Employee\_Audit table in your trigger.

Use the following code to test the trigger.

```
DELETE FROM Employee WHERE FirstName='Millie';
```

```
CREATE TRIGGER trgEmployeedeleted
```

```
ON EMPLOYEE
```

```
AFTER DELETE
```

```
AS
```



```
SET NOCOUNT ON;
```

```
INSERT INTO Employee_Audit (CreationTime, FirstName, City)
```

```
SELECT SYSDATETIME(), FirstName, City
```

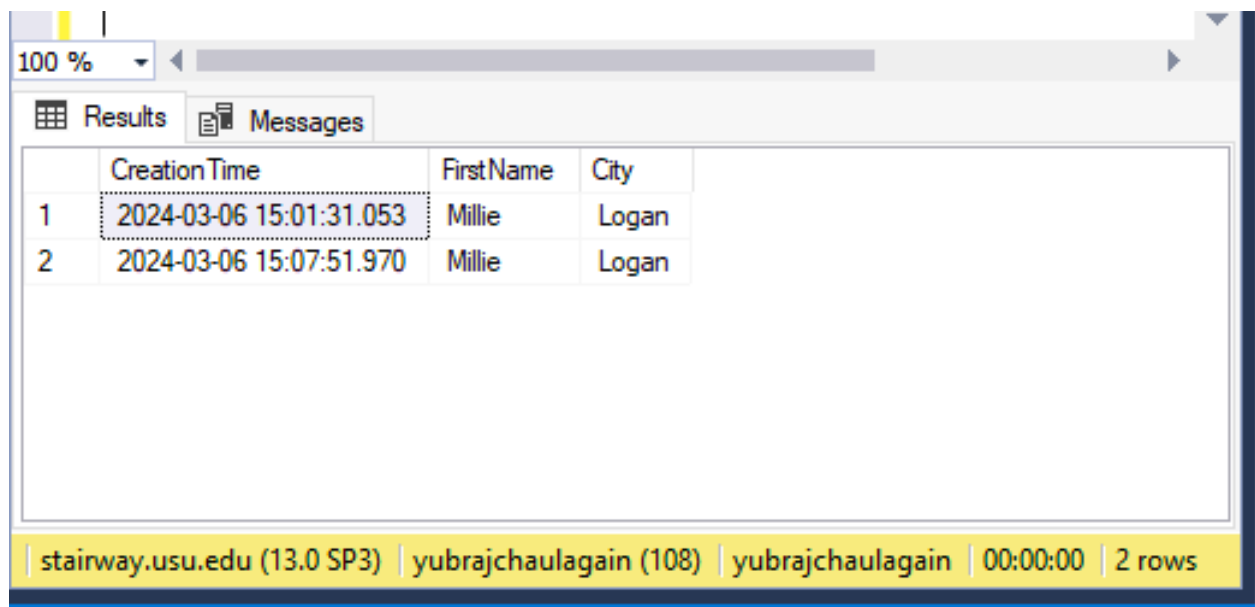
```
FROM deleted;
```

```
GO
```

```
DELETE FROM Employee WHERE FirstName='Millie';
```

```
SELECT * FROM Employee_Audit;
```

```
SELECT * FROM EMPLOYEE;
```



The screenshot shows a SQL Server Enterprise Manager window with a query results grid. The grid has four columns: CreationTime, FirstName, and City. There are two rows of data. The first row shows a creation time of 2024-03-06 15:01:31.053 for Millie in Logan. The second row shows a creation time of 2024-03-06 15:07:51.970 for Millie in Logan. The status bar at the bottom indicates the connection is to stairway.usu.edu (13.0 SP3), the user is yubrajchaulagain (108), the session is yubrajchaulagain, the time is 00:00:00, and there are 2 rows.

	CreationTime	FirstName	City
1	2024-03-06 15:01:31.053	Millie	Logan
2	2024-03-06 15:07:51.970	Millie	Logan

stairway.usu.edu (13.0 SP3) | yubrajchaulagain (108) | yubrajchaulagain | 00:00:00 | 2 rows