# CMPE 2400

Stored Procedures

# Passing Parameters by Name

- So far, when executing procedures, we have been passing our parameters by positions
  - i.e for each parameter, we only give the value of the parameter. The system will determine which value pertains to each parameter by the position in which the appears in the Exec statement
  - However, issues arise when some of the parameters have default values and we want to assign the default values to the columns.

# Lecture 12 Exercise 1

Create a procedure has a parameter for each of the non-identity columns of the employee table in your username_DB database. The order of the parameters should be first name, last name, marital status, HiredDate, hourly pay, and confirmation date.

Each time the procedure is executed, it adds a record to the employee table of your username_DB database, using the parameter values for each column.

Declare required variables, assign values to them and execute the procedure.

*If required, run the table creation script associated with this exercise.

# Lecture 12 Exercise 2

Modify the procedure in Exercise 1, so that the last 4 parameters have default values as below:

- **@marital char(1)='S',**
- **@hireDate date='2022-04-20',**
- **@hourlypay smallmoney=20,**
- **@confirmDate date='2022-07-01'**

Now execute the procedure with values provided for:

- All parameters
- The first 5 only
- The first 4 only
- The first 3 only
- The first 2 only

# Executing the procedure with Named Parameters

- In exercise 2, can we execute the procedure with default values for marital status and hire date but not for hourly pay and confirmation date?
- The way, we have executed our procedures so far, it's not possible. Since we are providing the parameters by positions, the ones using the default values always have to come at the end.
- The solution is to pass the parameter values by name.
- Each parameter is passed as @name=value,
  - **@name** is the name of the parameter as it appears in the definition of the procedure
  - **value** is either a literal or variable that has been assigned a value that we use in the execution of the procedure
  - E.g.

```
declare @Firstname varchar(30)='Jamie'
declare @LastName varchar(30)='Wattson'


exec SP_LE12Ex1 @FName=@FirstName, @LName= @LastName
```

# Executing the procedure with Named Parameters

With named parameters, we can give values for one or more of the default parameters, whatever be the order in which they appear.

Also with named parameters, we can execute the procedure with the parameter values being in a different order from that in which they appear in the procedure.

# Lecture 12 Exercise 3

- Execute the procedure from Exercise 2, providing default values for Marital Status and hired date, but non-default values for hourly pay and confirmation date.

# Using @@RowCount

- The @@RowCount Variable gives us the number of Rows affected during the execution of an SQL statement.

```
select EmployeeId, LastName, FirstName from NorthwindTraders.dbo.Employees

print 'Row Count is: ' + cast(@@Rowcount as varchar)
```

```
EmployeeId  LastName             FirstName
----------- -------------------- --------------------
1           Davolio              Nancy
2           Fuller               Andrew
3           Leverling            Janet
4           Peacock              Margaret
5           Buchanan             Steven
6           Suyama               Michael
7           King                 Robert
8           Callahan             Laura
9           Dodsworth            Anne
10          Pike                 Christopher

(10 rows affected)

Row Count is: 10
```

# Lecture 12 Exercise 4

- Write a stored procedure that has as parameter a first name, a last name, a hire date, an hourly pay, a confirmation date and a Marital Status with default value ‘S’. The procedure first checks if the first name and last name exist in the employee table (use @@RowCount). If it exists, the procedure returns 1 and doesn’t do anything else.  If the first name and last name combination doesn’t exist, the procedure attempts to insert a record with the provided values into the employee table. If the insert is successful, the procedure returns 2. Otherwise, it returns -1. Use try catch so that if something goes wrong the insert is not executed.
- Declare variables, assign values and execute the procedure. Obtain the error code and display a message so the user knows whether the insert was successful or not.
- If the insert was not successful (record existed already or error in execution), display a message to inform the user why the insert was not successful.

# Lecture 12 Exercise 5

- Modify the procedure in the previous exercise so that it:
  - Uses try catch
  - Has an output parameter that is set to the error message in case of an error in the execution of the statement (excluding the case where the record already existed). The default value of the output parameter can be ‘None’
- Execute the procedure so that it displays the messages as previously, but in the case of an error in the execution of the procedure (excluding existence of the record), it displays the reason for the error