CS 422 - DBMS

Lesson 8

Transact - SQL

Language of Nature

WHOLENESS OF THE LESSON

T-SQL is a programming language that allows SQL queries to be combined into an executable procedure.

Science & Technology of Consciousness: The language of nature is present everywhere, even in the language of a computer, but in a very restricted manner.

Need for T-SQL Over SQL

- SQL is a programming language that focuses on managing relational databases and is a common database language for all RDBMS products.
- SQL has its own limitations and so different RDBMS vendors have developed their own database language by extending SQL for their own RDBMS products.
- Microsoft added code to SQL and called it Transact-SQL or T-SQL. It’s the native language of SQL Server.
- Keep in mind that T-SQL is proprietary and is under the control of Microsoft while SQL, although developed by IBM, is already an open format.

Transact-SQL (T-SQL)

- T-SQL extends the SQL query language with a set of procedural commands.
- It includes variables, control structures, exception handling, etc.
- T-SQL batch commands can be used in expressions or packaged as stored procedures, user-defined functions or triggers.
  - A batch is a group of one or more T-SQL statements sent at the same time from an application to SQL Server for execution.
  - SQL Server compiles statements in a batch into an execution plan. The statements in the execution plan are then executed one at a time.
- The batch terminator, **GO**, can send the batch multiple times when followed by a number.

T-SQL Data Types

- T-SQL can use all SQL Server data types.
  - char(n): fixed length character data of length n.
  - varchar(n): variable-length character data where n is maximum length.
  - smallint: integers with range -32,768 to 32,767.
  - smallmoney: numbers up to $200,000.
  - float: ordinary floating point numbers.
  - smalldatetime: date and time values from Jan, 1900 to June, 2079.

Variables

- Variables are declared in the body of a batch or procedure with the DECLARE statement and are assigned values by using either a SET or SELECT statement.
- After declaration, all variables are initialized as NULL, unless a value is provided as part of the declaration.
- Variable names are not case sensitive.

Variables

```
DECLARE @Test int,
        @TestTwo char(20);
SET @Test = 1;
SET @TestTwo = 'Test Message';
Print @Test;
Print @TestTwo;
```

Output from the above is as follows:

```
1
Test Message.
```

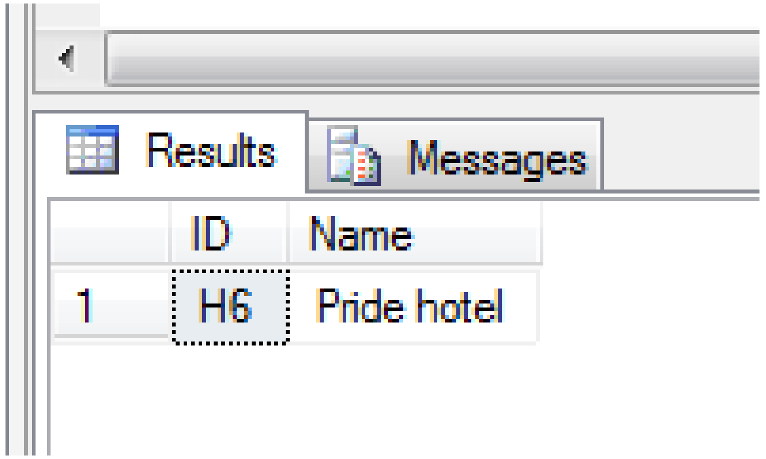
(The declared type of a variable may be any of the SQL Server data types.)

```
use hotelDB

DECLARE @TempID VARCHAR(4) = 99,
     @TempName VARCHAR(225) = '';

  SELECT @TempID = hotelNo,
         @TempName = hotelName
  FROM Hotel
  ORDER BY hotelNo;

SELECT @TempID AS ID, @TempName AS Name;
```

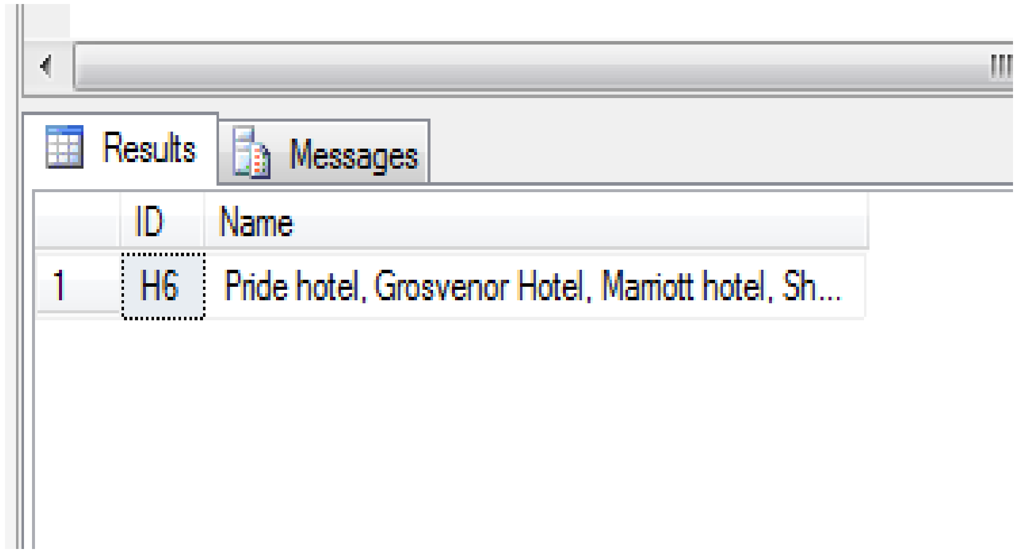


	ID	Name
1	H6	Pride hotel

Only the last row is stored in the variables *TempID* and *TempName*. **Never use a SELECT to populate a variable unless you're sure that it'll return only a single row.**

```
SELECT @TempName = @TempName + ',' + Hotel.hotelName
FROM Hotel;
```

Each row from Hotel table is appended to the variable TempName, changing the vertical column in the underlying table into a horizontal list.



	ID	Name
1	H6	Pride hotel, Grosvenor Hotel, Marriott hotel, Sh...

Conditionals

```
IF <Condition>
   <Statement>;

IF @Invar = 'Spectra'
  Print 'Input service is Spectra.'

IF <Condition>
  BEGIN
    <Multiple Statements>
  END;
```

CASE Statement

```
CASE column_name
      WHEN  condition1
THEN result1
      WHEN  condition2
THEN result2
      ...
      ELSE result
END
```

```
DECLARE @intInput int = 2

SELECT
  CASE(@intInput)
    WHEN 1 THEN 'One'
    WHEN 2 THEN 'Two'
    WHEN 3 THEN 'Three'
    ELSE 'Your message'
  END
AS testMsg;
```

CASE Statement Example

```
UPDATE Customer
   SET stateDesc =
      CASE statecode
           WHEN 'MA'
                THEN 'Massachusetts’
           WHEN 'VA'
                THEN 'Virginia’
           WHEN 'PA'
                THEN 'Pennsylvania’
           ELSE NULL
      END
```

Loops

```
DECLARE @count INT = 0;
WHILE @count < 3
  BEGIN
    Print @count;
    SET @count = @count + 1;
  END;
```

Output from the loop:

```
0
1
2
```

Error Handling

```
BEGIN TRY;
  SELECT * FROM Rates
  WHERE country = ‘Germany’;
  ...
END TRY
BEGIN CATCH
  PRINT ‘Error has been encountered.’;
  RETURN;
END CATCH;
```

Code inside the TRY block will be executed from beginning to end. If no errors occur, the CATCH block will be skipped. If an error occurs in the TRY block, execution will immediately jump to the CATCH block.

Stored Procedures

- A stored procedure is simply a compiled database object that contains one or more T-SQL statements.
- T-SQL stored procedures have input parameters, internal variables, output statements, conditionals, and looping statements.
- A stored procedure named `My_Proc` is executed by entering the following into the SQL Server Query window: `EXEC My_Proc`
- Comments are delimited by `/*` and `*/` as in C. (`--` can also be used for single line comment)

Advantages of Stored Procedures

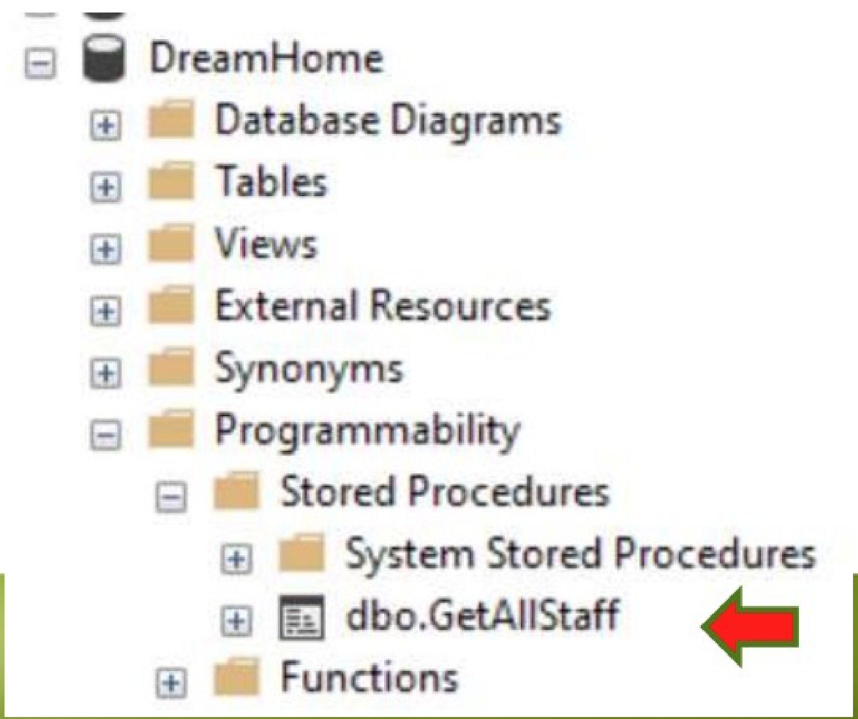
- **It can be easily modified in one place**
- **Reduced network traffic:** only the procedure name is passed over the network instead of the whole SQL code.
- **Reusable:** Stored procedures can be executed by multiple client applications without the need of writing the code again.
- **Security:** Stored procedures reduce the threat by eliminating direct access to the tables.
- **Performance:** The SQL Server stored procedure when executed for the first time creates a plan and stores it in the buffer pool so that the plan can be reused when it executes next time.

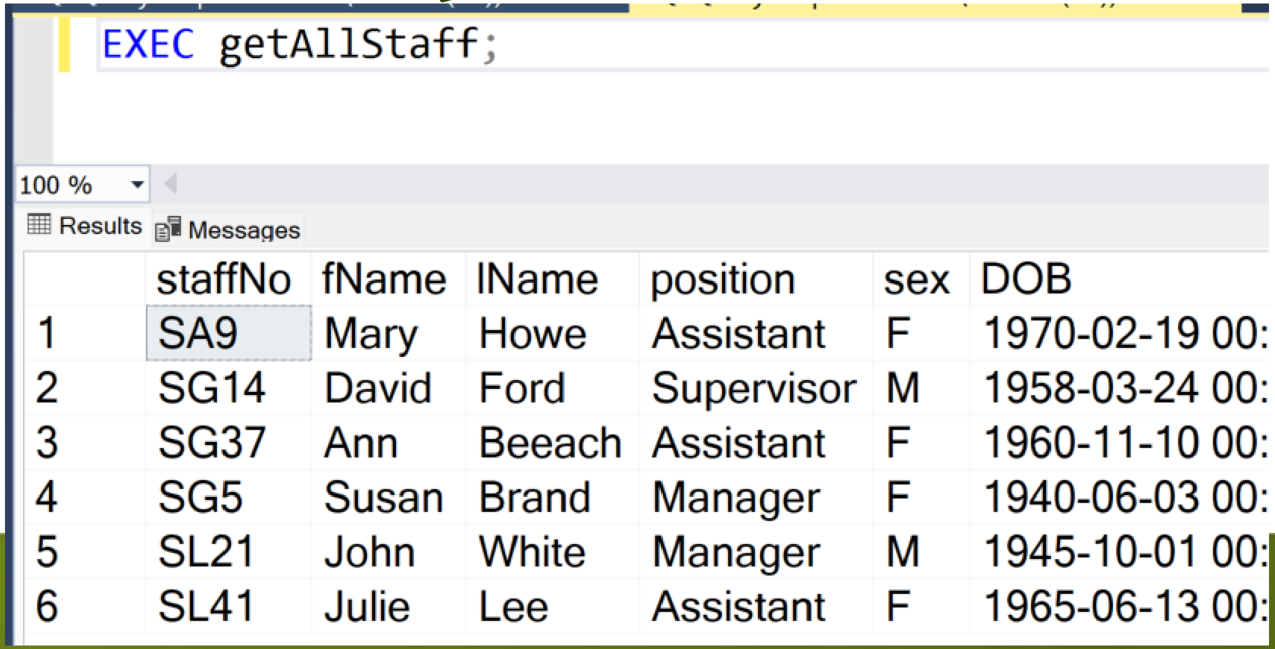
Create and Run a Stored Procedure

```
CREATE PROCEDURE GetAllStaff
AS
BEGIN
      select * from staff;
END
GO
```

After writing SP, Execute it (F5) to create a compiled SP object

After compiling, you can run the SP as shown below:



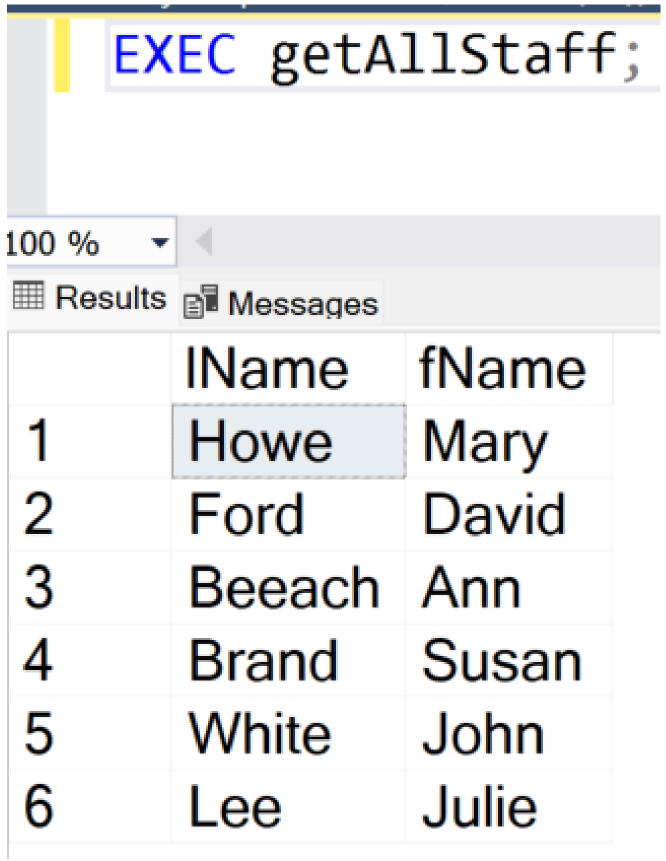


```
EXEC getAllStaff;
```

	staffNo	fName	lName	position	sex	DOB
1	SA9	Mary	Howe	Assistant	F	1970-02-19 00:
2	SG14	David	Ford	Supervisor	M	1958-03-24 00:
3	SG37	Ann	Beeach	Assistant	F	1960-11-10 00:
4	SG5	Susan	Brand	Manager	F	1940-06-03 00:
5	SL21	John	White	Manager	M	1945-10-01 00:
6	SL41	Julie	Lee	Assistant	F	1965-06-13 00:

Alter an Existing Stored Procedure

```
CREATE or ALTER PROCEDURE GetAllStaff
AS
BEGIN
    SELECT lName, fName FROM staff;
END
GO
```



```
EXEC getAllStaff;
```

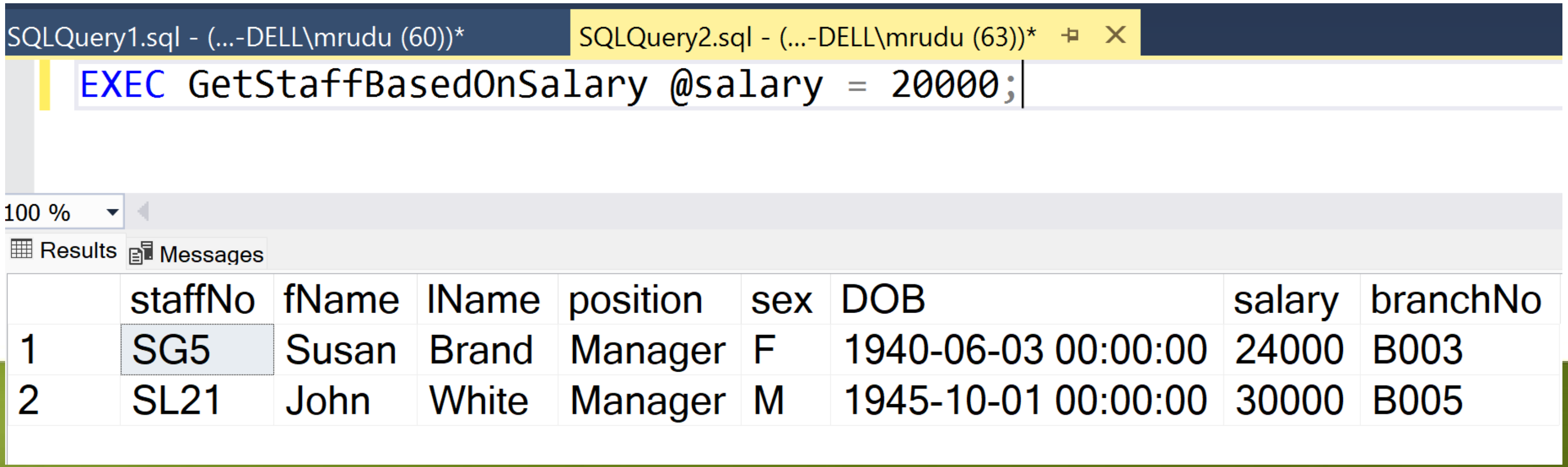
	lName	fName
1	Howe	Mary
2	Ford	David
3	Beeach	Ann
4	Brand	Susan
5	White	John
6	Lee	Julie

DROP Stored Procedure

```
DROP PROCEDURE GetAllStaff;
```

Stored Procedure with Parameter

```
CREATE OR ALTER PROCEDURE GetStaffBasedOnSalary
     (@salary int)
AS
BEGIN
     SELECT * FROM Staff WHERE salary >= @salary;
END
GO
```


SQLQuery1.sql - (...-DELL\mrudu (60))*
SQLQuery2.sql - (...-DELL\mrudu (63))*
100 %
Results
Messages

```
EXEC GetStaffBasedOnSalary @salary = 20000;
```

	staffNo	fName	lName	position	sex	DOB	salary	branchNo
1	SG5	Susan	Brand	Manager	F	1940-06-03 00:00:00	24000	B003
2	SL21	John	White	Manager	M	1945-10-01 00:00:00	30000	B005

Stored Procedure with Parameter contd..

```
CREATE PROCEDURE getCustomerDetails
          (@custName varchar(50))
AS
BEGIN
     SELECT * FROM Customer
     WHERE firstname = @custName;
END
```

To call this procedure from the SQL Server Query window:

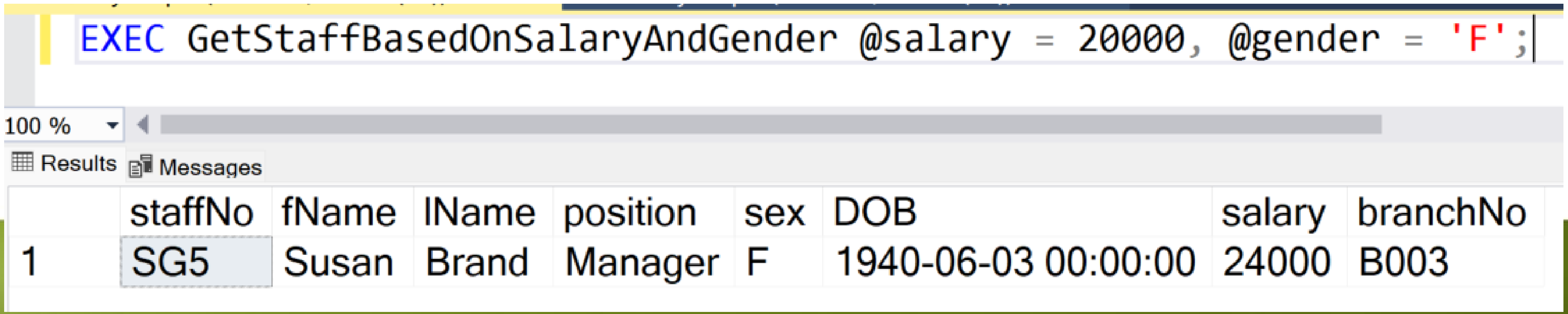
EXEC getCustomerDetails John

More Options

- **SET NOCOUNT ON**
  - The count (indicating the number of rows affected by a T-SQL statement) is not returned. When **SET NOCOUNT** is **OFF**, the count is returned. It is used with any SELECT, INSERT, UPDATE, DELETE statement.
- **SET QUOTED_IDENTIFIER ON/OFF**
  - When any character set that is defined in the single quotes ‘ ’ is treated as a literal.
- **SET ANSI_NULLS ON/OFF**
  - When it is set to OFF any comparison with NULL using = and <> will work as usual i.e. NULL = NULL returns true and 1 = NULL returns false.

Stored Procedure with Multiple Parameters

```
CREATE OR ALTER PROCEDURE GetStaffBasedOnSalaryAndGender
     @salary int,
     @gender varchar(1)
AS
BEGIN
     SELECT * FROM STAFF
     WHERE salary >= @salary AND sex = @gender
END
GO
```

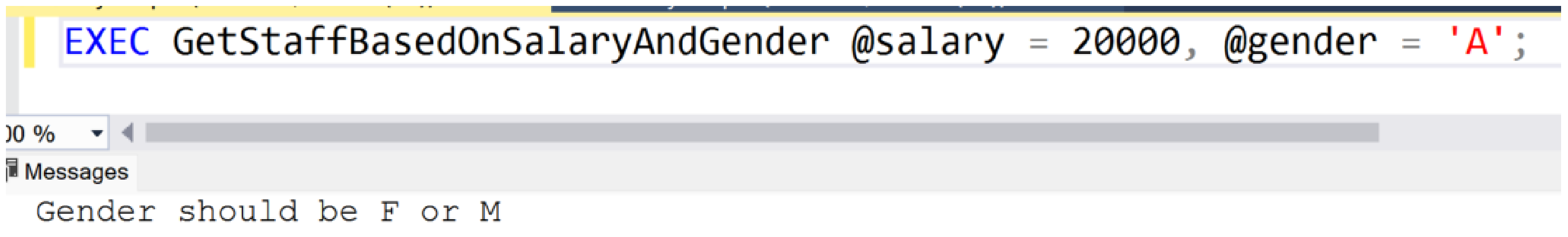


```
EXEC GetStaffBasedOnSalaryAndGender @salary = 20000, @gender = 'F';
```

	staffNo	fName	lName	position	sex	DOB	salary	branchNo
1	SG5	Susan	Brand	Manager	F	1940-06-03 00:00:00	24000	B003

User Input Validation Example 1

```sql
CREATE OR ALTER PROCEDURE GetStaffBasedOnSalaryAndGender
       @salary int,
       @gender varchar(1)
AS
IF @gender IN ('F', 'M')
    BEGIN
        SELECT * FROM STAFF
        WHERE salary >= @salary AND sex = @gender
    END
ELSE
    BEGIN
        PRINT 'Gender should be F or M'
        RETURN
    END
GO
```



```
EXEC GetStaffBasedOnSalaryAndGender @salary = 20000, @gender = 'A';
```

```
Gender should be F or M
```

User Input Validation Example 2

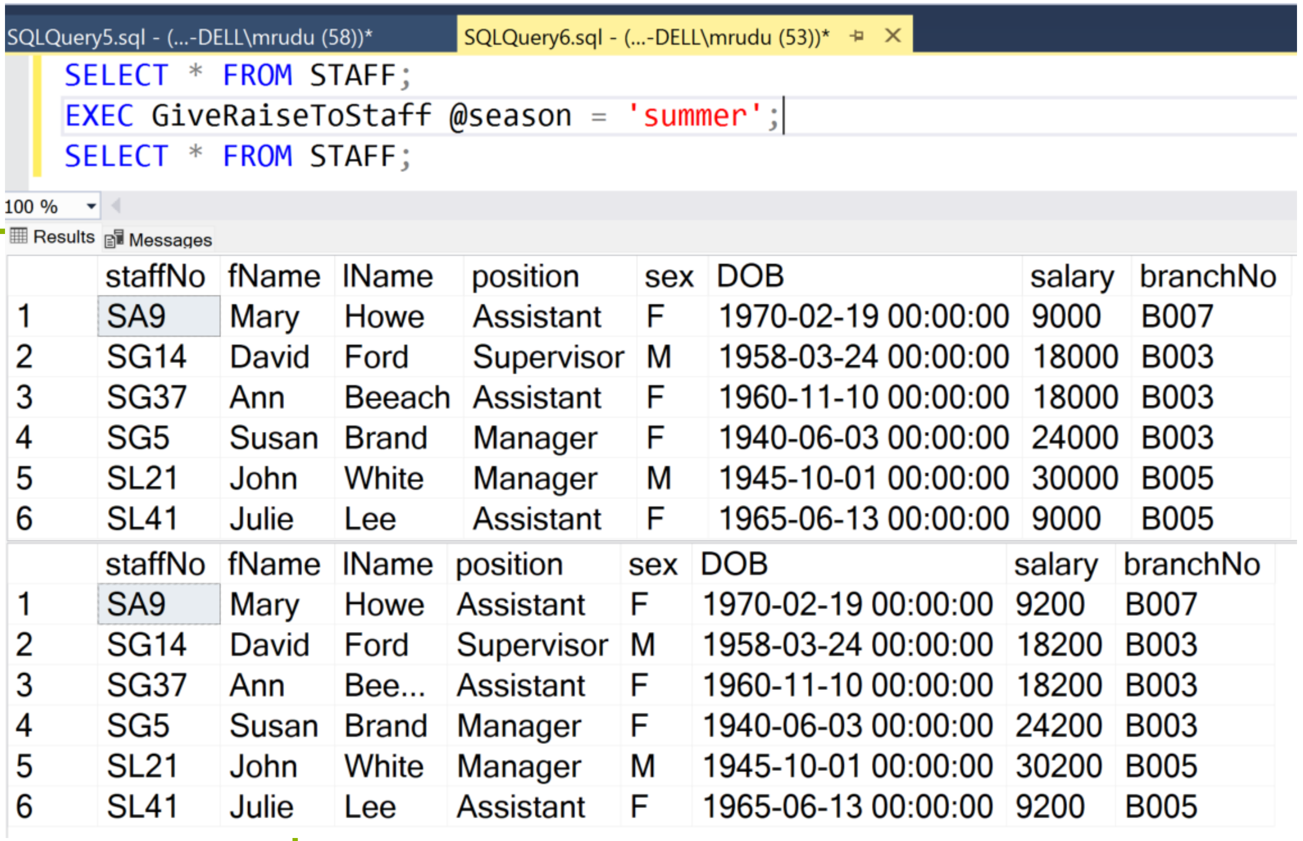
```
CREATE PROCEDURE GetRates(@country_name nchar(30))
AS
BEGIN
  IF EXISTS (SELECT * FROM Country_Table
             WHERE country = @country_name)
    BEGIN
      -- Find rates
    END
  ELSE BEGIN
    PRINT 'Error in country name.'
    RETURN
  END
END
```

EXISTS(<query>) is true if the query yields any rows at all.

Variables in SP

```
CREATE OR ALTER PROCEDURE GiveRaiseToStaff
        (@season varchar(10))
AS
BEGIN
        DECLARE @staffRaise int;
        -- compute raise
        IF @season = 'summer'
                SET @staffRaise = 200;
        IF @season = 'winter'
                SET @staffRaise = 400;

        -- upate salary
        UPDATE Staff SET salary = salary + @staffRaise
END
GO
```



SQLQuery5.sql - (...-DELL\mrudu (58))* SQLQuery6.sql - (...-DELL\mrudu (53))*

```
SELECT * FROM STAFF;
EXEC GiveRaiseToStaff @season = 'summer';
SELECT * FROM STAFF;
```

100 % Results Messages

	staffNo	fName	lName	position	sex	DOB	salary	branchNo
1	SA9	Mary	Howe	Assistant	F	1970-02-19 00:00:00	9000	B007
2	SG14	David	Ford	Supervisor	M	1958-03-24 00:00:00	18000	B003
3	SG37	Ann	Beeach	Assistant	F	1960-11-10 00:00:00	18000	B003
4	SG5	Susan	Brand	Manager	F	1940-06-03 00:00:00	24000	B003
5	SL21	John	White	Manager	M	1945-10-01 00:00:00	30000	B005
6	SL41	Julie	Lee	Assistant	F	1965-06-13 00:00:00	9000	B005

	staffNo	fName	lName	position	sex	DOB	salary	branchNo
1	SA9	Mary	Howe	Assistant	F	1970-02-19 00:00:00	9200	B007
2	SG14	David	Ford	Supervisor	M	1958-03-24 00:00:00	18200	B003
3	SG37	Ann	Bee...	Assistant	F	1960-11-10 00:00:00	18200	B003
4	SG5	Susan	Brand	Manager	F	1940-06-03 00:00:00	24200	B003
5	SL21	John	White	Manager	M	1945-10-01 00:00:00	30200	B005
6	SL41	Julie	Lee	Assistant	F	1965-06-13 00:00:00	9200	B005

```sql
IF (object_id('InsertEmployee')) is NOT NULL
                DROP PROCEDURE InsertEmployee
GO

CREATE PROCEDURE InsertEmployee
(
                @FirstName varchar(15),
                @LastName varchar(15),
                @Salary int,
                @HireDate datetime
)
AS
BEGIN
--Section 1: Define and initialize the local variable.
DECLARE @count int = 0

--Section 2: Determine whether the record already exists.
SELECT @count = COUNT(*)  FROM Employee
WHERE FirstName = @FirstName AND LastName = @LastName

--Section 3: Insert the record if it doesn't already exist.
IF (@count = 0)
      BEGIN
          INSERT INTO Employee VALUES
                (@FirstName, @LastName, @Salary, @HireDate)
          PRINT 'Employee record inserted'
       END
ELSE
       PRINT 'Employee record already exists...'
END
```

```sql
EXECUTE InsertEmployee
      @FirstName = 'Axel',
      @LastName = 'Brodie',
      @Salary = 145000,
      @HireDate = '2019-02-02'

EXECUTE InsertEmployee
      @FirstName = 'Pierre',
      @LastName = 'LaMontagne',
      @Salary = 135000,
      @HireDate = '2019-01-01'
```

Transactions

```
BEGIN TRY;

BEGIN TRANSACTION;
   INSERT INTO Student_Table
      SELECT * FROM Input_Students;
   DELETE FROM Input_Students;
COMMIT TRANSACTION;

END TRY
BEGIN CATCH
  /* error in the Insert-Delete sequence */
  ROLLBACK TRANSACTION;
  Print 'Processing of Student File has failed.';
END CATCH
```

Inserting into *Student_Table* and deleting from *Input_Students* must both be done for the database to maintain its integrity. The Transaction ensures that both will be done or neither will be done.

Problem 1

- Write and execute a T-SQL stored procedure *Factorial*(*n*), which computes and prints the factorial of the input parameter *n*.
- If *n* is negative, then the procedure prints an error message.
- E.g.
  - Command to use for executing the SP:
    ```
    EXEC getFactorial 5
    ```
  - Output should be:
    ```
    5! = 120
    ```

Problem 2

- The income tax is computed from the annual salary S and the number of dependents D.
- Net Salary: S - (7000 + D*950)
- Tax Computed as follows:
  - 10% of the first 15,000 of net salary;
  - plus 15% of the next 15,000 of net salary;
  - plus 28% of any remaining net salary over 30,000.

Problem 2 contd..

- Create a table Employee with the fields: SSN (PK), name, position, no. of dependents, salary.
- Write and execute a T-SQL stored procedure *Compute_Tax* to do the following:
    - create a new table *Tax* with fields: social security no., income tax.
    - fill the table *Tax* with data by computing the income tax for each person in the Employee Table.

UNITY CHART

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE:

T-SQL language for writing Stored Procedures

1. SQL queries can be used to view data in relations or update data in relations.
2. For complex activities that cannot be accomplished using a single SQL query. A T-SQL stored procedure allows many queries to be combined!

---

3. **Transcendental consciousness** is the experience of the simplest and most abstract state of awareness which underlies all states of greater excitation.
4. **Impulses within the Transcendental Field**: Transcendental consciousness has infinite energy, infinite creativity, and infinite intelligence, which allows the impulses within the transcendental field to create anything, giving it the qualities of infinite flexibility and infinite power.
5. **Wholeness moving within itself:** In unity consciousness one understands that all layers of nature are only different expressions of the same infinite field of pure consciousness.