User Defined Functions

User Defined functions can be used to perform a complex logic, can accept parameters and return data. SQL Server supports two types of User Defined Functions as mentioned below

Scalar Functions

The function which returns a Scalar/Single value.

```
CREATE FUNCTION MYSUM (@A INT, @B INT )
RETURNS INT
AS
BEGIN
        DECLARE @C AS INT;
        SET @C=@A+@B;
        RETURN @C;
END;

SELECT DBO.MYSUM(10,20);
DROP FUNCTION MYSUM;
```

Table Valued Functions

The function which returns a row set of SQL server Table.

```
CREATE FUNCTION GETEMP (
@DEP VARCHAR(50)
) RETURNS TABLE
AS
RETURN ( SELECT * FROM EMP WHERE DEPT = @DEP);
SELECT * FROM DBO.GETEMP('ADMIN')
```

Sub Queries

A Subquery or Inner query or Nested query is a query within another SQL query, and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause.
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- The BETWEEN operator cannot be used with a subquery; however, the BETWEEN can be used within the subquery.

Simple Subqueries

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows:

```
SELECT column_EID [, column_EID ]
FROM table1 [, table2 ]
WHERE column_EID OPERATOR
(SELECT column_EID [, column_EID ]
FROM table1 [, table2 ]
[WHERE])
```

Note: Although Subqueries are commonly used with Select statement, these can also be used with Insert, Update or Delete Statements

Correlated Subqueries

There are ways to incorporate the outer query's values into the subqueries clauses. These types of queries are called correlated subqueries, since the results from the subquery are connected, in some form, to values in the outer query. Correlated queries are sometimes called synchronized queries.

Eg:
Corelated sub query for average salary of those delhi employees whose salary is >100000

select avg(salary) from emp_sal
where eid in (select eid from emp where city='Delhi') and
eid in (select eid from emp_sal where salary > 100000);

Sql Exists Operator

• The EXISTS operator is used to test for the existence of any record in a subquery.
• The EXISTS operator returns true if the subquery returns one or more records.

```
SELECT column_EID(s)
FROM table_EID
WHERE EXISTS
(SELECT column_EID FROM table_EID WHERE condition);
```

Stored Procedures

- A stored procedure is prepared SQL code that we save so we can reuse the code over and over again. So if we think about a query that we write over and over again, instead of having to write that query each time we would save it as a stored procedure and then just call the stored procedure to execute the SQLcode that we saved as part of the stored procedure.
- In addition to running the same SQL code over and over again we also have the ability to pass parameters to the stored procedure.

SYNTAX

```
CREATE PROCEDURE <procedure_EID>
AS
BEGIN
<SQL Statement>
END

EXECUTE <procedure_EID>
EXEC <procedure_EID>
<procedure_EID>
```

Example 1 : Simple Procedure to get the details of Delhi employees

```
CREATE PROCEDURE SHDELEMP
AS
BEGIN
        SELECT * FROM EMP WHERE CITY = 'DELHI';
```

```
END;
```

Example 2 : Parameterized Procedure to get the details of employees of the specified city

```
CREATE PROCEDURE SHOWEMP @X VARCHAR(20)
AS
BEGIN
        SELECT * FROM EMP WHERE CITY = @X;
END;
```

Example 3 : Parameterized Procedure to get the contents of the specified table

```
CREATE PROCEDURE SHOW @Y VARCHAR(20)
AS
BEGIN
        EXEC('SELECT * FROM ' + @Y );
END;
```

Example 4 : Parameterized Procedure to insert the data in the emp_sal table

```
CREATE PROCEDURE IN_EMP_SAL
@ID VARCHAR(5), @A VARCHAR(20), @B VARCHAR(20), @X INT
AS
BEGIN
        SET NOCOUNT ON ;
        INSERT INTO EMP_SAL VALUES
        ( @ID, @A, @B, @X );

        SELECT * FROM EMP_SAL
        WHERE EID=@ID;
END;
```

A stored procedure with parameters:

```
SYNTAX
CREATE PROCEDURE <procedure_EID>
@ var1 datatype (size), var2 datatype (size)
AS
BEGIN
        [SET NOCOUNT ON/OFF]
        <SQL Statement>
END
```

EXECUTE <procedure_EID> var1 , var2
DROP PROCEDURE <procedure_EID>

Transactions

• A transaction is a unit of work that is performed against a database. For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing a transaction on the table.

Properties of Transactions

Transactions have the following four standard properties, usually referred to by the acronym ACID:

- Atomicity: Ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their former state.
- Consistency: Ensures that the database properly changes state upon a successfully committed transaction.
- Isolation: Enables transactions to operate independently of and transparent to each other.
- Durability: Ensures that the result or effect of a committed transaction persists in case of a system failure

There are following commands used to control transactions:

- COMMIT: To save the changes.
- ROLLBACK: To roll back the changes.
- SAVEPOINT: Creates points within groups of transactions in which to ROLLBACK.

Auto Increment

Auto Increment allows a unique number to be generated automatically when a new record is added to the table.

Identity (START, INCREMENT)

Example :
create table emp2

(id int identity (1,1) primary key,
EID varchar (30),
age int);