



The INSERT Statement



INSERT Statement

- Adds one or more new rows (records) to a table
- Syntax

```
INSERT [INTO] tableName
{
    [( column_list )]
    {
        | VALUES ( ( {DEFAULT | NULL | expression} [ ,...n ] )
        | derived_table
        | execute_statement
        | DEFAULT VALUES
    }
}
```



The Column List

- The COLUMN LIST is a comma-delimited list of column names enclosed in a parentheses
 - It provides a list of one or more columns in which to insert data
- If a column is not in the list of columns, then the column must:
 - Have an IDENTITY property
 - Have a DEFAULT
 - Allow NULL (nullable)
 - Be a COMPUTED column
- The Column List is optional ONLY IF
 - The values for all columns are supplied **and**
 - The values are given in the same order as the column names



The VALUES List

- The VALUES keyword specifies the values for one or more rows of a table
- The values list
 - Must be enclosed in **parentheses**.
 - Is specified as a comma-separated list of **scalar expressions**
 - Their data type, precision, and scale must be the same as the corresponding column in the column list
 - or implicitly convertible
- There must be one data value for each column name included in
 - The column-list or
 - The table



The VALUES List (cont.)

■ DEFAULT

- ☐ Used to force the Database Engine to load the default value defined for a column.
- ☐ If a default does not exist for the column and the column allows null values, **NULL** is inserted.
- ☐ DEFAULT is not valid for an identity column
- ☐ Use **DEFAULT VALUES** to insert the default values for all columns

■ Expression

- ☐ Must evaluate to a single value
- ☐ Should be of the same data type as the target column
- ☐ The Expression can be a constant, a variable, or a function



Examples

- specifying both the column names and the values to be inserted

```
INSERT INTO table_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...)
```

- Without specifying the column names

```
INSERT INTO table_name  
VALUES (value1, value2, value3, ...)
```

- Using DEFAULT and NULL

```
INSERT INTO table_name (column1, column2, column3, ...)  
VALUES (value1, DEFAULT, NULL, ...)
```



The Derived Table

- A Derived Table is any valid **SELECT** statement that returns rows of data
- The SELECT statement must provide a result set that is
 - In the **same order** as the columns in the table or in column-list
 - **Compatible** with the corresponding columns in the table or in column-list

- Example

INSERT INTO	<i>table2 (column1, column2, column3, ...)</i>
SELECT	<i>column1, column2, column3, ...</i>
FROM	<i>table1</i>
WHERE	condition



The Execute Statement

■ Execute Statement

- The EXECUTE statement is used to fire a stored procedure
- The stored procedure must return data (a result set) with a SELECT statement
- The result set should be
 - Compatible with the corresponding columns in column-list.
 - In the same order as the columns in the table or column-list

■ Example

```
INSERT Table1  
EXEC GetEmpNames
```

- **GetEmpNames** is a stored procedure that should return a value for **each column** of table *Table1* in the **same order**



SELECT INTO

- SELECT INTO is used to create a new table from values in another table
- It performs two actions
 - Creates a new table and
 - Populates it with the result set of a SELECT statement
- The structure of the new table is defined by the attributes of the expressions in the select list
- SELECT INTO can be used to combine data from several tables or views into one table



SELECT INTO - Example

- Copy only the customers in Germany into a new table

```
SELECT    CustomerName, TeleNumber  
INTO      CustomersGermany  
FROM      Customers  
WHERE     Country = 'Germany'
```



UNIQUEIDENTIFIER Data Type

- A column of **uniqueidentifier** data type can be used to identify a record throughout a database
 - It is unique throughout a database
- The value for a uniqueidentifier column can only be generated by the **NEWID()** or the **NEWSEQUENTIALID()** functions
- All column constraints and properties, except IDENTITY, can be used on the **uniqueidentifier** data type



UNIQUEIDENTIFIER - Example

```
CREATE TABLE myTable(  
    Col_1    UNIQUEIDENTIFIER DEFAULT newSequentialID()  
    , Col_2  VARCHAR(60)  
)  
GO
```

```
INSERT    myTable (Column_2)  
VALUES    ( 'test' )
```

```
INSERT    myTable (Column_2)  
VALUES    ( 'test1' )
```

Col_1	Col_2
64678BBC-40DF-E811-BE24-AC2B6E3DF454	test
65678BBC-40DF-E811-BE24-AC2B6E3DF454	test1



INSERT – Examples 1

- Using a Column-List and the VALUES clause

```
INSERT      MyTable ( myKey, Description )  
VALUES      ( 123 , 'A description of part 123' )
```

- Using SELECT INTO statement

```
SELECT      LastName, FirstName, Phone  
INTO        PhoneList  
FROM        Employee  
WHERE       DeptID = 2
```



INSERT – Examples 2

- Insert without column names

```
INSERT    Room
VALUES    ('Room 102', 45 )
```

- Insert using default values

```
INSERT    myTable
DEFAULT VALUES
```

- Inserting multiple rows with a single INSERT

```
INSERT    Department
VALUES    (1, 'Human Resources')
          , (2, 'Sales')
          , (3, 'Finance')
```



INSERT – Examples 3

- Using a SELECT subquery to insert one or more rows:

```
INSERT      TempTable (FName, LName)  
SELECT      FirstName, LastName  
FROM        Employee  
WHERE       DepID = 2
```

- Using EXECUTE to insert one or more rows:

```
INSERT      SalesStaff (StaffID, FullName)  
EXEC        spSalesStaff
```



INSERT – Examples 4

- Inserting data into a uniqueidentifier column
 - Uses the NEWID() function to obtain a unique value for column_2.

```
CREATE TABLE tblTest
(  
    column1 int IDENTITY,  
    column2 uniqueidentifier,  
)  
GO
```

```
INSERT    tblTest (column2)  
VALUES    ( NEWID() )
```