The INSERT Statement

The INSERT statement is used to add a new row or rows to a table. The basic syntax for the INSERT statement is shown here:

```
INSERT [INTO] table_name [(column_list)] VALUES
    expression | derived_table
```

The INTO keyword and the *column_list* parameter are optional. The *column_list* parameter specifies which columns you are inserting data into; these values will have a one-to-one correspondence (in order) with the values listed in the expression (which can be simply a list of values). Let's look at some examples.

Inserting Rows

The following code example demonstrates how to insert a single row of data into the *items* table:

```
INSERT INTO items
    (item_category, item_id, price, item_desc)
VALUES ('health food', 1, 4.00, 'tofu 6 oz.')
```

Because we specified a value for each column in the table and listed these values in the same order as their corresponding columns were defined in the table, we would not have to use the *column_list* parameter at all. But if the values were not in the same order as the columns, you could get the wrong data in a column or receive an error. For example, if you try to run this next statement, you will get the error message that follows it:

```
INSERT INTO items
VALUES (1, 'health food', 4.00, 'tofu 6 oz.')

Server: Msg 245, Level 16, State 1, Line 1
Syntax error converting the varchar value 'health food'
to a column of data type smallint.
```

This message is returned and the row is not inserted because we ordered the values incorrectly. We tried to insert the item ID into the *item_category* column, and the item category into the *item_id* column. The values were not compatible with the data types for those columns. If they had been compatible, SQL Server would have allowed us to insert the row, whether or not the values were in the proper place.

To see how the one row that we inserted in the table appears, query the table to select all rows, by using the following SELECT statement:

```
SELECT * from items
```

You will get the following result set:

```
item_category      item_id  price      item_desc
---------------   -------  --------   ------------
health food          1      4.00      tofu 6 oz.
```

When the *items* table was created, the *price* column was defined to allow null values, and the *item_desc* (description) column was assigned a default value of *No desc*. If no value is specified in the INSERT statement for the *price* column, a *NULL* will be inserted in that column for the new row. If no value is specified for the *item_desc* column, the default value *No desc* will be inserted in that column for the new row.

Omitting Column Values

In the first sample INSERT statement in the preceding section, we could have omitted values as well as column names for the columns *price* and *item_desc* because those columns have default values. If we omit a value for a column, we must specify the remaining columns in *column_list* because otherwise, SQL Server will match the listed values to the columns in the order in which the columns were defined in the table.

For example, suppose we leave out the *price* column value and do not specify any value for *column_list,* such as in this query:

```
INSERT  INTO  items
VALUES  ('junk  food',  2,  'fried  pork  skins')
```

SQL Server will attempt to insert the value given for *item_desc* (*fried pork skins*; the third value in the list of values) into the *price* column (the third column in the table). An error will result because *fried pork skins* is a *char* data type value, while *price* is a *smallmoney* data type. These are incompatible data types. The error message will look something like this:

```
Msg  213,  Level  16,  State  4,  Server  NTSERVER,  Line  1
Insert  Error:  Column  name  or  number  of  supplied  values
does  not  match  table  definition.
```

Imagine what could have happened to table integrity had *fried pork skins* been a value of a data type compatible with the data type specified for *price*. SQL Server would have unknowingly inserted the value into the wrong column, and the table data would have inconsistent data.

Remember that a value inserted in a table or view must be of a data type compatible with the column definition. Also, if a row being inserted violates a rule or constraint, you'll get an error message from SQL Server, and the insert will fail.

To avoid incompatible data type errors, order the names in *column_list* to match the order of the corresponding values, as shown here:

```
INSERT INTO items
    (item_category,  item_id,  item_desc)
VALUES  ('junk  food',  2,  'fried  pork  skins')
```

Because we did not specify the price, the *price* column will get a *NULL* for this row. Now execute the following SELECT statement:

```
SELECT  *  FROM  items
```

You should see the following result set (which now includes the two rows we inserted). Notice the *NULL* in the *price* column.

```
item_category      item_id  price         item_desc
---------------  -------  ---------  ----------------
health  food           1    4.00          tofu  6  oz.
junk  food             2    NULL          fried  pork  skins
```

Now let's add another row, without specifying values for either the *price* or the *item_desc* column, as shown here:

```
INSERT INTO items
       (item_category,  item_id)
VALUES  ('toys',  3)
```

The result set for this row alone can be found by using this query:

```
SELECT  *  FROM  items  WHERE  item_id  =  3
```

The result set will appear as follows:

```
item_category    item_id  price      item_desc
---------------  -------  ---------  -----------
toys              3          NULL      No  desc
```

Notice the *NULL* in the *price* column and the *No desc* in the *item_desc* column. You can change these values by using the UPDATE statement, as you'll see later.

The four types of columns for which SQL Server will automatically provide a value when one is not specified are columns that allow null values, columns with a default value, identity columns, and *timestamp* columns. We have seen what happens with *nullable* columns and columns with default values. An identity column gets the next available identity value, and a *timestamp* column gets the current timestamp value. In most cases, you cannot manually insert data values into these two types of columns.

NOTE

Exercise caution when performing an INSERT operation on a table. Be sure that the data you are inserting is being put in its intended column. Make sure you thoroughly test any T-SQL code before you use it to access or modify any important data.