Rawlston Morton

TSQL Ch11b HW

October 4, 2020

1. With SQL Server, you can construct a batch of T-SQL code as a character string and then execute that batch. This capability is called *dynamic SQL*. SQL Server provides two ways of executing dynamic SQL: using the *EXEC* (short for *EXECUTE*) command, and using the *sp\_executesql* stored procedure. I will explain the difference between the two and provide examples for using each.
2. The *sp\_executesql* stored procedure is an alternative tool to the *EXEC* command for executing dynamic SQL code. It’s more secure and more flexible in the sense that it has an interface; that is, it supports input and output parameters. Note that unlike *EXEC*, *sp\_executesql* supports only Unicode character strings as the input batch of code.
3. SQL injection, also known as SQLI, is a common attack vector that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed. This information may include any number of items, including sensitive company data, user lists or private customer details.
4. The *EXEC* command accepts a character string in parentheses as input and executes the batch of code within the character string. *EXEC* supports both regular and Unicode character strings as input. This command can also be used to execute a stored procedure, as I will demonstrate later in the chapter. The following example stores a character string with a *PRINT* statement in the variable *@sql* and then uses the *EXEC* command to invoke the batch of code stored within the variable: DECLARE @sql AS VARCHAR(100); SET @sql = 'PRINT ''This message was printed by a dynamic SQL batch.'';'; EXEC(@sql);
5. You can use input and output parameters in your dynamic SQL code can help you write more secure and more efficient code. In terms of security, parameters that appear in the code cannot be considered part of the code—they can only be considered operands in expressions. So, by using parameters, you can eliminate your exposure to SQL injection.
6. Routines are programmable objects that encapsulate code to calculate a result or to execute activity. SQL Server supports three types of routines: user-defined functions, stored procedures, and triggers.
7. Stored procedures are routines that encapsulate code. They can have input and output parameters, they can return result sets of queries, and they are allowed to have side effects. Not only can you modify data through stored procedures, you can also apply schema changes through them. A *trigger* is a special kind of stored procedure—one that cannot be executed explicitly. Instead, it’s attached to an event. Whenever the event takes place, the trigger fires and the trigger’s code runs. The purpose of a user-defined function (UDF) is to encapsulate logic that calculates something, possibly based on input parameters, and return a result.
8. A UDF is basically a method for SQL.
9. For example, invoking the *RAND* function to return a random value or the *NEWID* function to return a globally unique identifier (GUID) have side effects. Whenever you invoke the *RAND* function without specifying a seed, SQL Server generates a random seed that is based on the previous invocation of *RAND*. For this reason, SQL Server needs to store information internally whenever you invoke the *RAND* function. Similarly, whenever you invoke the *NEWID* function, the system needs to set some information aside to be taken into consideration in the next invocation of *NEWID*. Because *RAND* and *NEWID* have side effects, you’re not allowed to use them in your UDFs.
10. Procedure allows select as well as DML statement in it whereas function allows only select statement in it. Functions can be called from procedure whereas procedures cannot be called from function. ... UDF can be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section where as Stored procedures cannot be.