Abstract

Stored procedure and triggers have an irreplaceable importance in any database application, as they provide a powerful way to code application logic that can be stored on the server and execute according to the need of application. Writing stored procedures for database application involves set of sql statements with an assigned name that’s stored in the database in compiled form so that it can be shared by a number of programs. The use of stored procedures can be helpful in controlling access to data end-users may enter or change data but do not write procedures, preserving data integrity and improving productivity statements in a stored procedure only need to be written one time.

There are three types of sql statements

- DDL : define the database structure or schema.
- DML : procedural, non-procedural.
- TCL : security, integrity, concurrency control.

In any database application the data flow is controlled by sql statements, this gives us the flexibility to access data in particular form and in selected manner.

The stored procedures are the major part of any DB application, Stored procedures offer several distinct advantages over embedding queries in your Graphical User Interface (GUI).

II. Stored procedures and Triggers

In a database management system DBMS, a stored procedure is a set of Structured Query Language SQL statements with an assigned name that’s stored in the database in compiled form so that it can be shared by a number of programs. The use of stored procedures can be helpful in controlling access to data end-users may enter or change data but do not write procedures, preserving data integrity and improving productivity statements in a stored procedure only need to be written one time.

Stored procedures are similar to user defined functions (UDFs). The major difference is that UDFs can be used like any other expression within SQL statements, whereas stored procedures must be invoked using the CALL statement.
CALL <procedure name>(...) 
or
EXECUTE <procedure name>(...)

Stored procedures may return result sets, i.e., the results of a SELECT statement. Such result sets can be processed using cursors, by other stored procedures, by associating a result set locator, or by applications. Stored procedures may also contain declared variables for processing data and cursors that allow it to loop through multiple rows in a table.

Stored procedure languages typically include IF, WHILE, LOOP, REPEAT, and CASE statements, and more. Stored procedures can receive variables, return results or modify variables and return them, depending on how and where the variable is declared. Stored procedures are modular. This is a good thing from a maintenance standpoint. When query trouble arises in your application, you would likely agree that it is much easier to troubleshoot a stored procedure than an embedded query buried within many lines of GUI code. Stored procedures are tunable. By having procedures that handle the database work for your interface, you eliminate the need to modify the GUI source code to improve a query's performance. Changes can be made to the stored procedures—in terms of join methods, differing tables, etc.—that are transparent to the front-end interface.

Stored procedures abstract or separate server-side functions from the client-side. It is much easier to code a GUI application to call a procedure than to build a query through the GUI code.

Stored procedures are usually written by database developers/administrators. Persons holding these roles are usually more experienced in writing efficient queries and SQL statements.

### Triggers

The SQL CREATE TRIGGER statement provides a way for the database management system to actively control, monitor, and manage a group of tables whenever an insert, update, or delete operation is performed. The statements specified in the SQL trigger are executed each time an SQL insert, update, or delete operation is performed. An SQL trigger may call stored procedures or user-defined functions to perform additional processing when the trigger is executed. Unlike stored procedures, an SQL trigger cannot be directly called from an application. Instead, an SQL trigger is invoked by the database management system on the execution of a triggering insert, update, or delete operation.

The definition of the SQL trigger is stored in the database management system and is invoked by the database management system, when the SQL table, that the trigger is defined on, is modified. An SQL trigger can be created by specifying the CREATE TRIGGER SQL statement. The statements in the routine-body of the SQL trigger are transformed by SQL into a program object. The program is created in the schema specified by the trigger name qualifier. The specified trigger is registered in the SYSTRIGGERS, SYSTRIGDEP, SYSTRIGCOL, and SYSTRIGUPD SQL Catalogs. If SQL names were specified, the trigger will be created in the schema specified by the implicit or explicit qualifier. If system names were specified, the trigger will be created in the schema that is specified by the qualifier. If not qualified, the trigger will be created in the same schema as the subject table. If the trigger name is not a valid system name, or if a program with the same name already exists, the database manager will generate a system name.

NO CASCADE NO CASCADE is allowed for compatibility with other products and is not used by DB2 UDB for iSeries.

**BEFORE** Specifies that the trigger is a before trigger. The database manager executes the triggered-action before it applies any changes caused by an insert, delete, or update operation on the subject table. It also specifies that the triggered-action does not activate other triggers because the triggered-action of a before trigger cannot contain any updates.

**AFTER** Specifies that the trigger is an after trigger. The database manager executes the triggered-action after it applies any changes caused by an insert, delete, or update operation on the subject table. It also specifies that the triggered-action does not activate other triggers because the triggered-action of a BEFORE trigger cannot contain any updates.

**INSERT** Specifies that the trigger is an insert trigger. The database manager executes the triggered-action whenever there is an
**DELETE** Specifies that the trigger is a delete trigger. The database manager executes the triggered-action whenever there is a delete operation on the subject table.

A **DELETE** trigger cannot be added to a table with a referential constraint of ON DELETE CASCADE.

**UPDATE** Specifies that the trigger is an update trigger. The database manager executes the triggered-action whenever there is an update operation on the subject table.

An **UPDATE** trigger event cannot be added to a table with a referential constraint of ON DELETE SET NULL.

If an explicit column-name list is not specified, an update operation on any column of the subject table, including columns that are subsequently added with the ALTER TABLE statement, activates the triggered-action.

### III. Writing Stored procedures

It is not necessary that a stored procedure will have to written. It can be the case when a stored procedure doesn’t written anything. For Example, a stored procedure can be used to Insert, delete or update a sql statement.

Allows local variables, loops, procedures, examination of one tuple are at time.

**Rough Form**

```sql
DECLARE
  Declarations
BEGIN
  Executable statements
END
```

The DECLARE portion is optional

### IV. Preventing attacks in stored procedures

In this section, we present a stored procedure that is vulnerable to a SQLIA and explain how an attacker could exploit this vulnerability. We also present various techniques that can be employed to gain illegitimate access to the system as well as the network resources. A stored procedure is an operation set that is stored. Typically, stored procedures are written in SQL. Since stored procedures are stored on the server side, they are available to all clients. Once the stored procedure is modified, all clients automatically get the new version.

1. CREATE PROCEDURE [EMP].[RetrieveProfile] @Name varchar(50), @Passwd varchar(50) 
2. WITH EXECUTE AS CALLER
3. AS
4. BEGIN
5. DECLARE @SQL varchar(200);
6. ...
7. SET @SQL="select PROFILE from EMPLOYEE where ";
8. ...
9. IF LEN(@Name) > 0 AND LEN(@Passwd) > 0
10. BEGIN
11. ...
12. SELECT @SQL=@SQL+"NAME=""""+@Name+"" and ";
13. SELECT @SQL=@SQL+"PASSWD=""""+@Passwd+""""
14. ...
15. END
16. ELSE
17. BEGIN
18. ...
19. SELECT @SQL=@SQL+"NAME=""Guest""
20. ...
21. END
22. ...
EXEC(@SQL)
...
END

Code 1. Stored Procedure vulnerable to SQL-Injection

A sample stored procedure called with the username and password as user inputs in a variable length string format is shown in Code 1. Notice that, there is an EXEC system function which allow the user to dynamically build a SQL statement in string format and later execute it. This feature is supported in most other business database products also. Dynamically built SQL statements provide great user flexibility but also face a great threat from SQLIAs. The process of building an SQL statement could be used by the attacker to change the original intended semantics of the SQL statement. If the stored procedure in Code 1 is called with no values for @Name and @Passwd variables, the following query would get executed:

```
select PROFILE from EMPLOYEE where NAME='Guest'
```

When user inputs are provided for @Name and @Passwd, the following query would get executed:

```
select PROFILE from EMPLOYEE where NAME='name' and PASSWD='passwd'
```

In this scenario, suppose a user gives input for variable @Name as "" OR 1=1 −−" and any string, say "null", for the variable @Passwd the query would take the form: select PROFILE from EMPLOYEE where NAME="" or 1=1 −−" and PASS="null"

The characters "−−" mark the beginning of a comment in SQL, and everything after that is ignored. The query as interpreted by the database is a tautology and hence will always be satisfied, and the database would return information about all users. Thus an attacker can bypass all authentication modules in place and gain unrestricted access to critical data on the web server. An SQL-Injection Attack (SQLIA) is a subset of the unverified/unsanitized input vulnerability and occurs when an attacker attempts to change the logic, semantics or syntax of a legitimate SQL statement by inserting new SQL keywords or operators into the statement. This definition includes, but is not limited, to attacks based on tautologies, injected additional statements, exploiting untyped parameters, stored procedures, overly descriptive error messages, alternate encodings, length limits, second-order injections and injection of "UNION SELECT", "ORDER BY" and "HAVING" clauses. A detailed explanation of the different types and forms of SQLIAs and the ways in which they can be exploited are available in the public domain.

The widely deployed defense today is to train the programmers and web-developers about the security implications of their code and to teach them corrective measures and good programming practices. However, rewriting or revising the entire lot of existing legacy code is not an easy process and is not a financially viable option for many organizations. Even this does not guarantee any foolproof defense and hence we need automated processes to detect the vulnerability and eliminate them. Various other techniques like escaping the quotes and limiting the length of user inputs are employed as a quick fix solution. Unfortunately, even these security measures are only inadequate against highly sophisticated attacks. It is of even greater concern that well known database vendor products like Microsoft SQL Server etc. provide attackers direct access to the command line shell and registry using methods like xp cmdshell, xp regread etc. Some of the very recent incidents only highlight the magnitude of this problem and hence the urgent need to address it in an appropriate manner.

V. Stored procedures Vs Functions

Functions and stored procedures serve separate purposes. Although it's not the best analogy, functions can be viewed literally as any other function you'd use in any programming language, but stored procedures are more like individual programs or a batch script.

Functions normally have an output and optionally inputs. The output can then be used as the input to another function or as a predicate to a SQL Query – e.g., SELECT a, b, dbo.MyFunction(c) FROM table or SELECT a, b, c FROM table WHERE a = dbo.MyFunc(c).

A function is a subprogram written to perform certain computations

A scalar function returns only a single value (or NULL), whereas a table function returns a (relational) table comprising zero or more rows, each row with one or more columns.

Functions must return a value (using the RETURN keyword), but for stored procedures
this is not compulsory. Stored procedures can use RETURN keyword but without any value being passed.

Functions could be used in SELECT statements, provided they don’t do any data manipulation. However, procedures cannot be included in SELECT statements. A function can have only IN parameters, while stored procedures may have OUT or INOUT parameters. A stored procedure can return multiple values using the OUT parameter or return no value at all.

CREATE FUNCTION MyFunction (@someValue INTEGER) RETURNS INTEGER
AS
BEGIN
    DECLARE @retval INTEGER
    SELECT localValue
    FROM dbo.localToNationalMapTable
    WHERE nationalValue = @someValue

    RETURN @retval
END

VI. Conclusion

In the end we can say that a Stored procedure not only enhancing the possibility of reusing the code and execution plan, but it also increases the performance of the database by reducing the traffic of the network by reducing the amount of information send over the network. Design-time automation makes coding faster and ensures that all the procedures generated use the same naming conventions and structure. In an effort to improve their coding efficiency in a large SQL project, the authors wrote a set of design-time stored procedures that generate run-time stored procedures, and have used them in project after project ever since.

VII. References


[10] Dynamically maintain the teaching examples of triggers and stored procedures about the course of database application, Wu Da-sheng; Wu Sheng-yu Education Technology and Computer (ICETC), 2010 2nd International Conference