Executing Complex Statements

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

If you need to execute a complex statement that itself contains other statements in the SQL Commander, such as a CREATE PROCEDURE statement, you may need to help DbVisualizer figure out where the complex statement starts and ends. The reason is that DbVisualizer needs to send statements to the database for execution one by one.

To create or edit single procedures, functions, triggers, and similar types of complex objects, we recommend that you use the Create Procedure function and Procedure Editor to work with these.

The following explains the options that are available to run complex statements in the SQL Commander:

Using Execute Buffer

The **SQL->Execute Buffer** operation sends the complete editor buffer for execution as one statement. No comments are removed and no parsing of individual statements based on any delimiters is made. You can use this feature if the complex statement is the only statement in the SQL Commander editor.

Using an SQL Dialect

DbVisualizer will understand the syntax for complex statements for most of the officially supported databases. Open **Tools->Tool Properties->General->SQL Commander->Statement Delimiters** and check **Allow SQL Dialects** to enable this feature.

Here is an example of a complex statement for My SQL:

```
CREATE TRIGGER upd_check BEFORE UPDATE ON account
       FOR EACH ROW
       BEGIN
           IF NEW.amount < 0 THEN
               SET NEW.amount = 0;
           ELSEIF NEW.amount > 100 THEN
               SET NEW.amount = 100;
           END IF;
       END;
```

Note: Depending on the complexity of the dialect, the overhead of parsing and analyzing all statements before executing them may be significant. If you notice degraded performance when executing very large scripts we recommend that you disable the dialect feature and use some of the other methods to handle complex statements.

Using an SQL Block

To tell DbVisualizer that a part of a script should be handled as a single statement, you can insert an SQL block begin identifier just before the block and an end identifier after the block. The delimiter must be the only text on the line. The default value for the **Begin Identifier** is `--/` and for the **End Identifier** i t is `/`.

Here is an example of an SQL block for Oracle:

```
--/
script to disable foreign keys

declare cursor tabs is select table_name, constraint_name
  from user_constraints where constraint_type = 'R' and owner = user;

begin
  for j in tabs loop
    execute immediate ('alter table '||j.table_name||' disable constraint'||j.constraint_name);
end loop;
end;
/
```

Using the @delimiter Command

With the **@delimiter** command you can temporarily change the statement delimiter DbVisualizer uses to separate the statements and send them one by one to the database. Use it before the complex statement, and after the statement if the script contains additional statements. Here's an example:

```
@delimiter ++;
CREATE OR REPLACE FUNCTION HELLO (p1 IN VARCHAR2) RETURN VARCHAR2
AS
BEGIN
 RETURN 'Hello ' || p1;
END;
++
@delimiter ;++
@call ${returnValue||(null)||String||noshow dir=out}$ = HELLO('World');
@echo returnValue = ${returnValue}$;
```

The first @delimiter command sets the delimiter to ++ so that the default `;` delimiter can be used within the function body in the CREATE statement. The ++ delimiter is then used to end the CREATE statement, and another **@delimiter** command sets the delimiter back to `;` for the remaining commands in the script.

Calling a Function or Procedure

As a general rule, DbVisualizer takes no part in how you execute the query; the SQL code is simply sent to the server for interpretation and execution. You should be able to write and execute the SQL code just as you do in any other tool, subject to the server's ability to understand it.

However, the command for executing a function or procedure varies among vendors. For instance, **EXECUTE** runs a procedure in Microsoft Transact-SQL and Oracle SQL*Plus, whereas **CALL** does the same thing in PostgreSQL and MySQL.

@call is the way to do it in DbVisualizer. This is also transparent to the underlying database; you use the same command irrespective of the database you are connected to (see Executing a Code Object and Using Client-Side Commands for more details).