system.db.runPrepUpdate

This function is used in **Python Scripting.**

Description

Runs a prepared statement against the database, returning the number of rows that were affected. Prepared statements differ from regular queries in that they can use a special placeholder, the question-mark character (?) in the query where any dynamic arguments would go, and then use an array of values to provide real information for those arguments. Make sure that the length of your argument array matches the number of question-mark placeholders in your query. This call should be used for UPDATE, INSERT, and DELETE queries.

This is extremely useful for two purposes:

- This method avoids the problematic technique of concatenating user input inside of a query, which can lead to syntax errors, or worse, a nasty security problem called a SQL injection attack . For example, if you have a user-supplied string that is used in a WHERE clause, you use single-quotes to enclose the string to make the query valid. What happens in the user has a single-quote in their text? Your query will fail. Prepared statements are immune to this problem.
- This is the only way to write an INSERT or UPDATE query that has binary or BLOB data. Using BLOBs can be very hand for storing images or reports in the database, where all clients have access to them.

The "?" placeholder refers to variables of the query statement that help the statement return the correct information. The "?" placeholder cannot reference column names, table names, or the underlying syntax of the query. This is because the SQL standard for handling the "?" placeholder excludes these items.

Client Permission Restrictions

Permission Type: Legacy Database Access

Client access to this scripting function is blocked to users that do not meet the role/zone requirements for the above permission type. This function is unaffected when run in the Gateway scope.

Syntax

system.db. runPrepUpdate(query, args, [database], [tx], [getKey], [skipAudit])

- Parameters

  String query - A query (typically an UPDATE, INSERT, or DELETE) to run as a prepared statement with placeholders (?) denoting where the arguments go.

  Object[] args - A list of arguments. Will be used in order to match each placeholder (?) found in the query.

  String database - Optional, The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.

  String tx - Optional, A transaction identifier. If omitted, the update will be executed in its own transaction.

  Boolean getKey - Optional, A flag indicating whether or not the result should be the number of rows returned (getKey=0) or the newly generated key value that was created as a result of the update (getKey=1). Not all databases support automatic retrieval of generated keys.

  Boolean skipAudit - Optional, A flag which, if set to true, will cause the prep update to skip the audit system. Useful for some queries that have fields which won't fit into the audit log.

- Returns

  Integer - The number of rows affected by the query, or the key value that was generated, depending on the value of the getKey flag.

- Scope

  All

Code Examples

Code Snippet

```
# This example would gather some user entered text and insert it into the database.

userText = event.source.parent.getComponent("TextArea").text
userName = system.security.getUsername()
system.db.runPrepUpdate("INSERT INTO Comments (Name, UserComment) VALUES (?,?)", [userName,
userText])
```

Code Snippet

```
# This example would gather some user entered text and insert it into the database.
# The difference between this example and the previous example is that this example is
explicitly declaring which database connection to run the query againt.
# Sometimes you need to run a query against a database connection that is not the default
connection.

userText = event.source.parent.getComponent("TextArea").text
userName = system.security.getUsername()
databaseConnection = "AlternateDatabase"
system.db.runPrepUpdate("INSERT INTO Comments (Name, UserComment) VALUES (?,?)", [userName,
userText], databaseConnection)
```

Code Snippet

```
# This code would read a file and upload it to the database

filename = system.file.openFile() # Ask the user to open a file
if filename != None:
   filedata = system.file.readFileAsBytes(filename)
   system.db.runPrepUpdate("INSERT INTO Files (file_data) VALUES (?)", [filedata])
```

Code Snippet

```
# This example inserts a new user and gives it the 'admin' role. Demonstrates the ability to
retrieve a newly created key value.

# Get the username/password
name = event.source.parent.getComponent('Name').text
desc = event.source.parent.getComponent('Description').text
building = event.source.parent.getComponent('Building').selectedValue

# Insert the value
id = system.db.runPrepUpdate("INSERT INTO machines (machine_name, description) VALUES (?,
?)", [name, desc], getKey=1)

# Add a row to the user role mapping table
system.db.runPrepUpdate("INSERT INTO machine_building_mapping (machine_id, building) VALUES
(?, ?)", [id, building])
```

Code Snippet - Inserting Data From a Table Component

```
# This example will take a dataset from a table component, and insert new records into the
database, one row at a time


# Read the contents of the table
tableData = event.source.parent.getComponent('Table').data

# Convert it to a PyDataset. This is mostly for convenience, as they're easier to iterate
through
pyData = system.dataset.toPyDataSet(tableData)

# Build the query we'll use. You could easily modify the line to accommodate the table you're
trying to insert into.
query = "INSERT INTO my_table (col1, col2) VALUES (?, ?)"

# Iterate
for row in pyData:

        # Build an arguments list based on the current row. Using indexing here, so 'row[0]'
is the 1st column, 'row[1]' is the 2nd column, etc
        args = [row[0], row[1]]

        # Add a row to the database. You could optionally check the contents of the row
first, and add an if-statement to prevent the record based on some criteria
        system.db.runPrepUpdate(query, args)
```