Queries in Scripting

Overview

In addition to using a binding, queries can be called from a Python script. This typically involves using one of Ignition's built-in scripting functions. This page presents several approaches to interacting with a database from a Python Script. See the Scripting section for more information about using Python.

Named Queries

Named Queries may be called from a Python script using the system.db.runNamedQuery function. Named Queries can execute as a Prepared Statement when not utilizing Query String parameters, so they can offer a secure approach to calling a query from a script.

Additionally, Named Queries are incredibly easy to call from a script, since you only need to know their name, and don't have to bother with SQL syntax in the script. This is another way to protect your database from other users in the Designer, as you can Protect Named Queries.

Python - Calling a Named Query

```
namedQuery = "Add New Order"
parameters = {"accountId":123, "productName":"Bananas"}

system.db.runNamedQuery(namedQuery, parameters)
```

For more information, refer to the Named Queries section.

Prepared Statements vs Standard Statements

A prepared statement is a feature in a database that executes a parameterized query. One of the main reasons to utilize a prepared statement is because they are resilient to SQL Injection Attacks. Because of this, we **highly recommend** you utilize Prepared Statements over Standard Statements, but Named Queries are the most secure. Especially in cases where the users can type values that will be included in the query.

Prepared Statements typically involve passing two pieces of information to the database:

- A query string that contains placeholders to represent where arguments will be passed in later. These are represented in the query as question mark characters ("?")
- A series of arguments to replace the placeholder characters in the static query.

Standard Statements vs Prepared Statements

Typical SQL insert queries look like the following:

SQL - Standard Statement

```
INSERT INTO orders (account_id, product_name) VALUES (123, 'Bananas')
```

A Prepared Statement instead looks like the following. Note, that the placeholders do not require quotation marks even when a string will be passed in.

SQL - Prepared Statement

```
INSERT INTO orders (account_id, product_name) VALUES (?, ?)
```

How do I Call a Prepared Statement?

Prepared Statements can be called from a script using specific functions. Typically, they contain "Prep" in the name such as system.db.runPrepQuery, or system.db.runPrepUpdate. When in doubt, take a look at the sub pages in the system.db section.

There are typically two required parameters with these functions: a string literal that represents the query, and a list of parameters.

Python - Calling an Insert Query as a Prepared Statement

```
query = "INSERT INTO orders (account_id, product_name) VALUES (?, ?)"
args = [123, "Bananas"]
system.db.runPrepUpdate(query, args)
```

Notable Prepared Statement Functions

- system.db.runPrepQuery: Should be used when SELECT statements are used, as it returns a dataset containing the results. This function is effectively "read-only", as it does not manipulate data in the database.
- system.db.runPrepUpdate: Should be used when manipulating the database in some way. When executing a statement that utilize INSERT, UPDATE, or DELETE, the runPrepUpdate function should be called. Note that it does return the number of rows affected, so the return value can be used to keep track of how much of an impact the query had.
- system.db.runScalarPrepQuery: Like runPrepQuery above, but only returns the first column of the first row: i.e. a single value is returned instead of a full dataset. This is useful when using an aggregate function of some sort to return a count or total, as it saves your script the work of extracting the value from the full dataset that runPrepQuery normally returns.

Standard Statements

Queries can be called as a Standard Statement (a statement that that isn't a Prepared Statement) by using the system.db.runQuery and system.db.runUpdateQuery functions. However, these are susceptible to **SQL Injection attacks**, and should be avoided where possible: especially when users have access to a keyboard and can directly type values that will be used in the query.

Calling a Standard Statement involves building the entire query as a single string, and passing the string on to our Standard Statement functions.

Python - Calling an Insert Query as a Standard Statement

```
query = "INSERT INTO orders (account_id, product_name) VALUES (%i, '%s')" % (123, "Bananas")
system.db.run(query)
```

Notable Standard Statement Functions

- system.db.runQuery: Executes a SELECT statement, returning a result set as a dataset.
- system.db.runUpdateQuery: Executes a statement that manipulates the database in someway. Should be used with INSERT, UPDATE, and DELETE statements.
- system.db.runScalarPrepQuery: Similar to runQuery, except only a single value is returned: the first column of the first row. Generally used in conjunction with SELECT statements that contain an aggregate function.

Stored Procedures

If your database administrator has already configured Stored Procedures for you to use, then they can easily be called from a Python Script. Using Stored Procedures in a script typically involves two main steps:

1. A SProcCall object is created with the system.db.createSProcCall function. The SProcCall object contains several functions that can be used to register parameters, and access the results set returned by the Stored Procedure after it has been executed.
2. The system.db.execSProcCall function must be used to execute the Stored Procedure.

Python - Creating and Executing a Stored Procedure

```
# Create a SProcCall object, which will be used to configure parameters on the Stored Procedure, and
then executed.
myCall = system.db.createSProcCall("insert_new_order")

# Register parameters on the SProcCall object.
myCall.registerInParam(1, system.db.INTEGER, 123)
myCall.registerInParam(2, system.db.VARCHAR, "Bananas")

# Execute the Stored Procedure.
system.db.execSProcCall(myCall)
```

Take a look at the SQL Stored Procedures page for more details.

Transactions

A SQL Transaction can also be executed from a script. For the unfamiliar, a Transaction is a batch of statements that will be executed together, and either succeed or fail as a group. Note, that the statements executed in the Transaction are not visible by other connections in the database until you commit them.

Transactions typically involve several steps:

1. Call system.db.beginTransaction. This returns a **transaction identifier** that can be used with other statements. Using this identifier is how you specify that a statement should be included in the transaction.
2. Start calling other statements with other functions, such as system.db.runPrepUpdate. The function's "tx" parameter will be passed the **transaction identifier**.
3. Commit or Rollback the transaction. Use system.db.commitTransaction to commit, and system.db.rollbackTransaction to rollback. These options are essentially the same as applying or canceling the results of the queries. Commiting will make the updated results available to other database connections.
4. Close the Transaction once you're done, which can be accomplished with the system.db.closeTransaction function. This invalidates the **transaction identifier**.

Python - The SQL Transaction Workflow

```
# 1) Begin the transaction. This returns a transaction identifier that we can use with other statements.
transactionId = system.db.beginTransaction(timeout = 5000)

# 2) Now we can execute statements. Because we want these to run as part of the transaction, we need to
include our identifier.
query = "INSERT INTO orders (account_id, product_name) VALUES (?, ?)"
args = [123, "Bananas"]
system.db.runPrepUpdate(query, args, tx = transactionId)

# 3) We can continue to add statements, but in this case we'll commit them. We could instead rollback if
there was an issue with our previous statement.
system.db.commitTransactions(transactionId)

# 4) We're done, so close the Transaction.
system.db.closeTransaction(transactionId)
```

Related Topics ...

- Writing Basic SQL Queries
- system.db.runPrepUpdate
- Named Queries
- SQL Stored Procedures