Using GlideRecord to query tables

London

Subscribe •••

In order to query a table, first create an object for the table.

This object is called a GlideRecord. To create a GlideRecord, create the following in script:

```
var target = new GlideRecord('incident');
```

This creates a variable called target which is a GlideRecord object for the incident table. The only parameter needed is the name of the table to be accessed.

To process all records from the incident table, add the following script:

```
target.query(); // Issue the query to the database to get all records
while (target.next()) {
  // add code here to process the incident record
}
```

This issues the `query()` to the database. Each call to `next()` would load the next record which you would process and do whatever you want to do.

But that is not the common case. Most of the time you actually want to retrieve a specific record or a specific set of records, and you have some criteria (query conditions) that define the records you want to obtain. For example, say you want to obtain all the incident records that have a priority value of 1. Here is the code that would accomplish that.

```
var target = new GlideRecord('incident');
target.addQuery('priority',1);
target.query(); // Issue the query to the database to get relevant reco
rds
while (target.next()) {
  // add code here to process the incident record
}
```

Notice in the above code we added the line `target.addQuery('priority', 1);`. This is indicating that you only want the records where the priority field is equal to 1. We assume that the majority of queries that you will want to do will be equality queries, queries where you want to find records where a field is equal to a value. Therefore we provide this format of the query and do not make you specify that you want an equals operation, we just assume it. However, lets say you wanted to find all incidents where the priority field is GREATER THAN 1. In that case you have to provide us with the operator that you want to apply to priority and this is done by providing the operator in the `addQuery()` request as is shown below.

```
var target = new GlideRecord('incident') ;
target.addQuery('priority','>',1);
target.query(); // Issue the query to the database to get relevant reco
rds
while (target.next()) {
  // add code here to process the incident record
}
```

Available JavaScript operators

Describes the operators that can be used within an `addQuery()` request.

Available JavaScript Operators

=
Field must be equal to value supplied.
addQuery('priority', '=', 1);
>
Field must be greater than value supplied.
addQuery('priority', '>', 1);
<
Field must be less than value supplied.
addQuery('priority', '<', 3);
>=
Field must be equal or greater than value supplied.
addQuery('priority', '>=', 1);

<=	Field must be equal or less than value supplied.	addQuery('priority', '<=', 3);
!=	Field must not equal the value supplied.	addQuery('priority', '!=', 1);
STARTSWITH	Field must start with the value supplied. The example shown on the right returns all records where the *short_description* field starts with the text Error.	addQuery('short_description', 'STARTSWITH', 'Error');
CONTAINS	Field must contain the value supplied somewhere in the text. The example shown on the right returns all records where the *short_description* field contains the text Error anywhere in the field.	addQuery('short_description', 'CONTAINS', 'Error');
IN	Takes a map of values that allows commas, and gathers a collection of records that meet some other requirement. Behaves as `Select * from <table> where short_description IN ('Error', 'Success', 'Failure')` , which is identical to `Select * from <table> where short_description='Error'` . For example, to query all variable values that belong to a specific Activity, use the IN clause, and store their sys_ids in a map, or comma-separated list. Then query the variable value table and supply this list of sys_ids.	addQuery('short_description', 'IN', 'Error,Success,Failure');
ENDSWITH	Field must terminate with the value supplied. The example shown on the right returns all records where the *short_description* field ends with text Error.	addQuery('short_description', 'ENDSWITH', 'Error');

DOES NOT CONTAIN
Selects records that do NOT match the pattern in the field. This operator does not retrieve empty fields. For empty values, use the operators "is empty" or "is not empty."The example shown on the right returns all records where the *short_description* field does not have the word "Error."
addQuery('short_description', 'DOES NOT CONTAIN', 'Error');
NOT IN
Takes a map of values that allows commas, and gathers a collection of records that meet some other requirement. Behaves as: `Select * from <table> where short_description NOT IN ('Error')`.
addQuery('short_description', 'NOT IN', 'Error,Success,Failure');
Special operator that retrieves only records of a specified "class" for extended tables. The code example on the right, shows how to retrieve all configuration items that are classified as computers.
addQuery('sys_class_name', 'INSTANCEOF', 'cmdb_ci_computer');

For additional information on the operators that are available for filters and queries, see Operators available for filters and queries.

There are also some special methods that you can use to search for data that is NULL or NOT NULL. To search for all incidents where the *short_description* field has not been supplied (is null), use the following query:

```
var target = new GlideRecord('incident');
target.addNullQuery('short_description');
target.query(); // Issue the query to the database to get all records
while (target.next()) {
  // add code here to process the incident record
}
```

To find all incidents in which a *short_description* has been supplied, use the following query:

```
var target = new GlideRecord('incident');
target.addNotNullQuery('short_description');
target.query(); // Issue the query to the database to get all records
while (target.next()) {
  // add code here to process the incident record
}
```

For more information on the `GlideRecord` API and its available methods, see GlideRecord .

GlideRecord query examples

These examples demonstrate how to perform various `GlideRecord` queries.

query

```
var rec = new GlideRecord('incident');
rec.query();
while(rec.next()) {
  gs.print(rec.number + ' exists'); }
```

update

```
var rec = new GlideRecord('incident');
rec.addQuery('active',true);
rec.query();
while(rec.next()) {
  rec.active = false;
  gs.print('Active incident ' + rec.number = ' closed');
  rec.update(); }
```

insert

```
var rec = new GlideRecord('incident');
rec.initialize();
rec.short_description = 'Network problem';
rec.caller_id.setDisplayValue('Joe Employee');
rec.insert();
```

delete

```
var rec = new GlideRecord('incident');
rec.addQuery('active',false);
rec.query();
while(rec.next()) {
  gs.print('Inactive incident ' + rec.number + ' deleted');
  rec.deleteRecord(); }
```

Querying Service Catalog Tables

You cannot directly query the variables of the Service Catalog Request Item table [sc_req_item]. Instead, query the Variable Ownership table, [sc_item_option_mtom], by adding two queries, one for the variable name and another for the value. The query returns the many-to-many relationship, which you can dot-walk to the requested item. The following example finds the request items that have the variable *item_name* with a value of *item_value* and displays the request item numbers:

```
var gr = new GlideRecord('sc_item_option_mtom');
gr.addQuery('sc_item_option.item_option_new.name','item_name');
gr.addQuery('sc_item_option.value','item_value');
gr.query();

while(gr.next()) {
  gs.addInfoMessage(gr.request_item.number); }
```

For additional information see GlideRecord .

Last Updated: July 26, 2018

Tags: London, Now Platform Custom Business Applications, Scripting

Send Feedback

Previous Topic Next Topic