Using Table Filters

Servoy provides high-level filtering functionality which may be applied to any database table using the addTableFilterParam method of the databaseManager API. Table filters have the following properties:

- **Scope** - A filter may be applied to a single database table or, if no table is specified, an entire server connection. A filter will constrain the records which are returned by any queries that are issued from the Servoy client to the table(s). A filter takes effect immediately upon being added and remains in effect for the duration of the client session unless programmatically removed. The constraints of a filter apply to all facets of a Servoy solution, including:
  - foundsets
  - related foundsets
  - value lists

There are only two ways to circumvent a table filter:
by issuing a custom SQL query String through the getDataSetByQuery method of the databaseManager API. (Note: the version of getDataSetByQuery that takes a QueryBuilder object as first parameter DOES take into account applicable TableFilters

by using the rawSQL plugin

Therefore, if one wishes to maintain the effects of a filter, it is important to remember to modify queries with an appropriate SQL WHERE clause in case any of the above two cases apply

- **Logical Expression** - A filter will contain a logical expression which is evaluated on behalf of records in the filtered table(s). Only records, for which the expression evaluates to true, will be returned by any queries issued to the filtered table(s). At runtime, the filter will be translated into an SQL WHERE clauseand appended to the query of any foundset which is bound to the filtered table(s). An expression contains the following components:
  - Data Provider Name - This is the left-hand operand. It is the name of a single column by which to filter. When filtering an entire server connection, only tables which contain the named column will be filtered.
  - Operator - The following operators are supported

=	Only records whose column **equals** the specified value
<	Only records whose column is **less than** the specified value
>	Only records whose column **greater than** the specified value
>=	Only records whose column **greater than or equals** the specified value
<=	Only records whose column **less than or equals**the specified value
!=	Only records whose column does **not equal** the specified value
^	Only records whose column value **is null**
LIKE	Only records whose column matches using the **SQL LIKE** construct (use wildcard '%' characters)
IN	Only records whose column value is in (using the SQL IN construct) a list of values
BETWEEN	Only records whose column value is (inclusive) between a list of 2 values
#	Modifier, used to make case-insensitive queries
||	Modifier used to concatenate two conditions w/ a logical OR

Operators and modifiers may be combined, producing more complex conditions. For example #^||!= would translate to: is null OR case-insensitive not equals

  - Data Provider Value - This is the right-hand operand and should evaluate to a literal value to be compared with the named column.

When using the IN operator, one should provide an array of values or a String, which may be used as a sub select for the SQL IN clause.

- **Filter Name** - When adding a table filter parameter, a filter name may be used to allow for the later removal of a named filter. Multiple parameters or conditions may be set using the same filter name. In this case, all parameters may be removed at the same time.

Example This is a simple example which filters records in a products table based on the criterion that the status is not discontinued

```
var success = databaseManager.addTableFilterParam('crm_server','products','product_status','!=',globals.
STATUS_DISCONTINUED,'productfilter');
```

Example This example shows a two filters using the **IN** operator

```
// Filter products within an array of product codes
var success = databaseManager.addTableFilterParam('crm', 'products', 'productcode', 'in', [120, 144, 200]);

// Filter orders using a subselect
var success = databaseManager.addTableFilterParam('crm', 'orders', 'countrycode', 'sql:in', 'select country code
from countries where region = "Europe"')
```

In Servoy 2020.03 we added support for sql-modifier . When value of the filter is a custom query, the operator used should be prefixed with "sql:" (like sql:in , sql:=, ...). This is a security feature, so a developer should explicitly mark a value as a select query. Currently, the operator works without the sql-modifier (the old way), but it will generate a warning in the log: "Filter is created using a custom query without using the sql-modifier, this will be removed in a future version of servoy, please use operator 'sql:in'". See issue

SVY-14682 - Jira project doesn't exist or you don't have permission to view it.

as we plan to enforce the modifier in future releases.

Example This example shows filters on null values

```
// Filter products within product code is null
var success = databaseManager.addTableFilterParam('crm', 'products', 'productcode', '=', null);

// Filter products within product code is not null
var success = databaseManager.addTableFilterParam('crm', 'products', 'productcode', '!=', null);

// Filter products within product code is null or 120
var success = databaseManager.addTableFilterParam('crm', 'products', 'productcode', '^||=', 120);
```

Example This example shows how to filter an entire server connection by passing <null> for the table name. This is ideal for multi-tenant architectures as an entire server connection can be filtered by a single expression, i.e. the current company

```
// all tables that have the companyid column should be filtered
var success = databaseManager.addTableFilterParam('crm', null, 'companyidid', '=', globals.currentCompanyID)
```