

Topic updated on September 21, 2023

Local DB Select

The **Local DB Select** action retrieves rows from a Local Database table. An optional Where clause and Group By clause can be used.

The SELECT parameter can be an asterisk (*), which is the default, to indicate select all columns, or the SELECT parameter can be any valid SQL select statement.

When all columns are selected, the Logical type of the Output variables will be the data type from the table definition.

When the **Return Result as Array** parameter is set to **True**, the Output tab variables are treated as arrays, the Output **Row Index** variable is not used, and the SELECT parameter is not available (it is handled as a SELECT *).

1.Local DB Select

SELECT:

*

FROM:

TABLE_2_96

WHERE:

GROUP BY:

Max Rows:

10

Return Result as Array:

False

Output

Routing

Details

Output

Name	Logical	Count	Value	Type
C04	FLOAT8	1		
C03	INT8	1		
C02	INT8	1		
C01	STRING(32)	1		
Error Message	STRING(128)	1		
Rows Selected	INT4	1		
Row Index	INT4	1		

When a select statement other than the default SELECT * is used, the Logical type of the Output variables will be ANY.

1.Local DB Select

SELECT:

C01, C02 as Column_2

FROM:

TABLE_2_96

WHERE:

GROUP BY:

Max Rows:

10

Return Result as Array:

False

Output

Routing




Details

Output

Name	Logical	Count	Value	Type
C01	ANY	1		
Column_2	ANY	1		
Error Message	STRING(128)	1		
Rows Selected	INT4	1		
Row Index	INT4	1		

Parameter descriptions

You must supply valid SQL statements within the **Select** and **Where** clause. The use of String Format Specifiers may cause runtime problems.

Parameter	Description
SELECT	<p>The Select clause is used to determine the Action's Output. The FROM parameter (the Local Database table) must be selected and contain data for validation to occur.</p> <ul style="list-style-type: none">This can be an asterisk (*), to indicate all columns. An empty (blank) select parameter is treated the same as an asterisk.A valid SQL statement, including alias names for columnsAll SQLite aggregate functions are supported, see the SQLite Aggregate Functions List. <p>The select clause builder, accessed by selecting the icon , can be used to assist in building the select clause.</p>
FROM	<p>The name of the Local Database table to select the rows from. The table name will be available from a drop-down list.</p>
WHERE	<p>A Where clause can be used to restrict the rows selected. Both constants and substitution variables can be used in the Where clause. To construct a Where clause, use an operator (=, !=, >, >=, <, <=, like, is null, is not null) to relate the column to either a constant or a substitution variable. Each of these operators can be combined with other operators using an And or Or statement. To use a substitution variable, insert \$(x) where x is a variable that will be associated to a variable on the Input tab (see the Input tab below).</p> <p>Note: For columns of type TEXT, the constant or substitution variable must be enclosed in double quotes. For example:</p> <ul style="list-style-type: none">C01 = "JohnDoe" The constant is enclosed in double quotes.C01 = "\$ (test)" The substitution variable is enclosed in double quotes. The variable (test in this example) will be added to the Input tab and can then be mapped to a STRING variable (or a constant). <p>The where clause builder, accessed by selecting the icon , can be used to assist in building the where clause.</p>
GROUP BY	<p>A Group by clause can be used to to collect data across multiple rows and group the results by one or more columns.</p> <p>The group by clause builder, accessed by selecting the icon , can be used to assist in building the group by clause.</p>
Max Rows	<p>The maximum number of rows the query is expected to return. The query can return less rows than the Max Rows specified. However, if the query returns more rows than specified in Max Rows, only the number of rows specified by Max Rows will be actually written to the output variables. The other rows will be discarded.</p>
Return Result as Array	<p>An option to return the results as an array. True - Returns the results as an array, up to the Max Rows value. The Output tab variables will have a Count parameter equal to the Max Rows and the output variables will be treated as arrays. The Row Index variable is not used. False - Returns the results one row at a time. The Output tab variable Row Index is returned as an index variable. Routing is controlled by the Routing tab Next Row parameter.</p>

Select clause builder

The SELECT parameter can be an asterisk (*), which is the default, to indicate select all columns.

An example of a Local DB Select action with the SELECT and WHERE parameters specified is:

1.Local DB Select

SELECT:

SUM(C03)

...

FROM:

TABLE_2_96

WHERE:

C01 = "\$(test)"

...

GROUP BY:

...

Max Rows:

10

Return Result as Array:

False

Input

Output

Routing

Details

Input

Name	Logical	Count	Value	Type
test	ANY	1	globVar.strings[1]	STRING(30)

?

The select clause builder can be used to assist in building the select clause.

When the **Return Result as Array** parameter is set to **True**, the SELECT parameter is not available (it is handled as a SELECT *).

The select clause builder allows selections of columns, aggregate functions and alias names. For example:

Table: TABLE_2_96

SELECT SUM(C03)

Column	Function	Alias
C03	SUM	

Buttons: Add All, Add, Delete, Up, Down, OK, Cancel

The output of the select clause builder is placed in the SELECT parameter and can be manually modified.

- The Add All, Add, Delete, Up and Down buttons handle the placement of the table column names in the select clause.
- The optional aggregate functions (none, AVG, Count, MAX, MIN, SUM) are selected in the **Function** pull down column.
- An alias name can be assigned to each output of the select clause in the **Alias** column, and will be used as the variable name on the Output tab.
- The where clause (see next section) can be used to restrict the rows selected.

Example select clauses:

Examples of the select clause as it is displayed in the select clause builder include:

- SELECT *
Select all rows, return the value of all columns.
- SELECT C03
Select all rows, return the value in the column named C03.
- SELECT SUM(C03)
Select all rows, return the Sum of the values of the column named C03.
- SELECT SUM(C03) AS C03_sum
Select all rows, return the Sum of the values of the column named C03 in an output variable named C03_sum.
- SELECT SUM(C03) AS C03_sum, AVG(C03) AS C03_avg, MAX(C03) AS C03_max
Similar to the previous example, returns the Sum, Avg and Max of the values of the column named C03 in the output variables: C03_sum, C03_avg, C03_max.

Where clause builder

The where clause builder can be used to assist in building the where clause. For example:

Database Columns: C04, C03, C02, C01

Operators: =, !=, >, >=

And (selected), Or

Add

Statements: C01 = "\${test}"

Move Up, Move Down, Remove

OK, Cancel

The output of the where clause builder is placed in the WHERE parameter and can be manually modified.

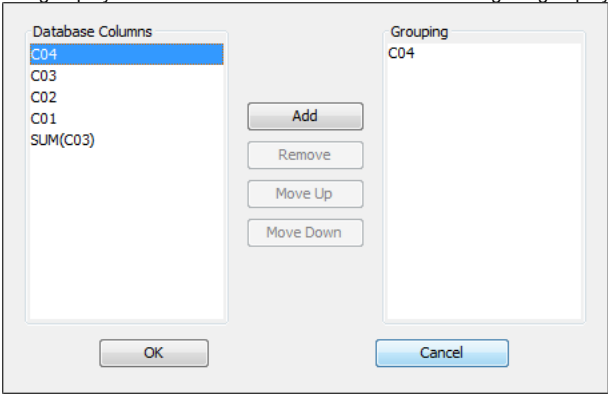
- The where clause builder displays all columns from the Local DB table specified in the **FROM** parameter.
- To build a where clause: Select a column name from the **Database Columns** list, select an operator from the **Operators** list, and select **And** or **Or** (used when adding multiple columns to the where clause). Then select the **Add** button.
- The Move Up, Move Down and Remove buttons handle the placement of the column names in the where clause.
- Substitution variables can be used to indicate a variable should be specified in the input tab

Example where clauses:

- WHERE C01 = "\$ (test)"
Restrict the select rows to those with a value for the column named C01 to be equal to the value of the substitution variable **test**.
The substitution variable **test** will be inserted as a row in the input tab. It can then reference any available variable in the system (device variable, trigger variable, constant, etc.).
- WHERE C02 >= 10
Restrict the selected rows to those with a value for the column named C02 to be greater than or equal to 10.
- Where C01 = "\$ (test)" AND C02 >= 10
Restrict the select rows to those that meet the criteria for C01 and C02.
- **Note:** For columns of type TEXT, the constant or substitution variable must be enclosed in double quotes. For example:
 - C01 = "JohnDoe"
The constant is enclosed in double quotes.
 - C01 = "\$ (test)"
The substitution variable is enclosed in double quotes. The variable (test in this example) will be added to the Input tab and can then be mapped to a STRING variable (or a constant).

Group by clause builder

The group by clause builder can be used to assist in building the group by clause. For example:



The output of the group by clause builder is placed in the GROUP BY parameter and can be manually modified.

Select and Where clause changes

After using the select and where clause builders, manual changes to the SELECT and WHERE parameters may result in invalid SQL statements. Validation of the parameters will be attempted and a warning displayed if there is a problem parsing the statement.

Input tab

Optional. The **Input** tab will only appear when a Where clause is specified and the Where clause contains a substitution variable (for example, \$ (test)).

There will be as many rows in the **Input** tab as there are substitution variables in the Where clause.

Parameter	Description
Logical Variables from the Where clause	Required. The variable whose value is to be substituted in the Where clause statement. This variable can then reference any available variable in the system (device variable, trigger variable, constant, etc.)

Output tab

The **Output** tab will have one row per returned variable (such as a column name), based on the select clause.

Parameter	Description
Output Name	The returned variable. This could be a column name, the result of an aggregate function, or an alias name. The value of the returned variable is written to the variable specified in the Value cell. Important: if a returned value is a null and that value is mapped to a numeric variable, the trigger will fail because a null cannot be written to a numeric field.
Error Message	Optional. Used to provide information if the SQL query fails.
Rows Selected	The number of rows returned by a SQL query operation.
Row Index	The current row number when iterating through the rows returned by the Select operation, when the Return Result as Array parameter is set to False . When iterating through the rows returned by the Local DB Select action, Row Index is incremented by one for each row processed. Using the Next Row route from the Routing tab, it is possible to iterate through each row returned by the Local DB Select action. For each row processed, Row Index is incremented by 1. For more information, see Example using Local DB Select action .

Tip

Next Row Routing

When the **Return Result as Array** is set to **False**, the returned rows must be processed in a loop. The actions in the row processing loop are indicated by the execution path starting at the **Next Row** routing option. For more information, see [Example using Local DB Select action](#).

Routing tab

On Result	Description
Success	The action completed successfully.
Failure	The action encountered a failure.
Next Row	Fetch the next row of values retrieved from the select statement when the Return Result as Array parameter is set to False .

Related topics

- [Local Database](#)
- [Transaction](#)
- [Local DB Export](#)