MSSQL Connector

The **MSSQL Connector** allows you to perform CRUD operation on MSSQL database. You can choose the required operation from the dropdown using templates from your BPMN process.

Prerequisites

To start working with the **MSSQL Connector**, a relevant database user password must be configured and stored as a secret in your cluster. The user must have permission to perform database operation on given database instance.

Create a MSSQL Connector task

Currently, the MSSQL Connector supports seven types of operations: create database, create table, insert data into the table, delete data from the table, update table data, read table data and alter table.

To use a **MSSQL Connector** in your process, either change the type of existing task by clicking on it and using the wrench-shaped **Change type** context menu icon or create a new Connector task by using the **Append Connector** context menu. Follow our guide on using Connectors to learn more.

Make your MSSQL Connector executable

To make the **MSSQL Connector** executable, fill out the mandatory fields highlighted in red in the properties panel.

Database connection Object input for MSSQL Connector

MSSQL Connector database connection object takes – host, port, username and password.
e.g. localhost, 1433, username, password (as secrets Token e.g. secrets.MSSQL_TOKEN)

Create a new database

MS SQL DATABASE CONNECTOR
CREATE DATABASE

General

Template Applied

Details

Operation

Operation
Create Database
Operation to be done

Database Connection

host
= hostname
Hostname/computer name or IP address of mssql server host. e.g. localhost

port
= 1433
Port for MS SQL server. e.g. 1433

username
= username
Username with DB access

password
= secrets.MS_SQL
Password for username e.g. secrets.TOKEN, Secrets can be used to reference encrypted authentication credentials in Connectors. See the Secrets documentation for more details.

Input Mapping

databaseName
= databaseName
New database name

Output Mapping

Result Variable
createDatabaseStatus
Name of variable to store the response in

Result Expression
=
Expression to map the response into process variables

Error Handling

Error Expression
=
Expression to handle errors. Details in the documentation.

To create a database, take the following steps:

1. In the **Operation** section, set the field value **Operation** as **Create Database**.
2. Set the required parameters and credentials in the **Database Connection** section. See the relevant appendix entry to find out more.
3. In the **Input** Mapping section, set the field **databaseName** as the desired name of a database you wish to create. For example, `MyNewDatabase`.
   Alternatively, you could use a FEEL expression.

Create Database operation response

You can use an output mapping to map the response:

1. Use **Result Variable** to store the response in a process variable. For example, `createDatabaseStatus`.

Create a new table

MS SQL DATABASE CONNECTOR
CREATE TABLE

Details

General

Template Applied

Operation

Operation
Create Table
Operation to be done

Database Connection

host
= hostname
Hostname/computer name or IP address of mssql server host. e.g. localhost

port
= 1433
Port for MS SQL server. e.g. 1433

username
= username
Username with DB access

password
= secrets.MS_SQL
Password for username e.g. secrets.TOKEN, Secrets can be used to reference encrypted authentication credentials in Connectors. See the Secrets documentation for more details.

Input Mapping

databaseName
= databaseName
Name of the database for new table

tableName
= tableName
New table name

columnsList

```
= [
    {
        "colName": "MEMBER_ID",
        "DataType": "bigint",
        "Constraints": [
          "PRIMARY KEY",
            "NOT NULL"
        ]
    },
    {
        "colName": "memberName",
        "DataType": "varchar(50)",
        "Constraints": "NOT NULL"
    },
    {
        "colName": "Sys_date",
        "DataType": "datetime"
    }
]
```

List of the columns in format: [{'colName':'Age', 'dataType': 'varchar(255)', 'constraints': ['UNIQUE']}]

Output Mapping

Result Variable
createTableStatus
Name of variable to store the response in

Result Expression
=
Expression to map the response into process variables

Error Handling

Error Expression
=
Expression to handle errors. Details in the documentation.

To create a table, take the following steps:

1. In the **Operation** section, set the field value **Operation** as **Create Table**.
2. Set the required parameters and credentials in the **Database Connection** section. See the relevant appendix entry to find out more.
3. In the **Input Mapping** section, set the field **databaseName, tableName** as the desired name of a table you wish to create. For example, `MyNewTable`. Alternatively, you could use a FEEL expression.
4. Set **columnsList**, using FEEL expression as List of columns details, which is a List of context having keys as colName, datatype and constraints.

Create Table operation response

You can use an output mapping to map the response:

1. Use **Result Variable** to store the response in a process variable. For example, `createTableStatus`.

Insert data into the table

MS SQL DATABASE CONNECTOR
INSERT DATA

Details

General

Template Applied

Operation

Operation
Insert Data
Operation to be done

Database Connection

host
= hostname
Hostname/computer name or IP address of mssql server host. e.g. localhost

port
= 1433
Port for MS SQL server. e.g. 1433

username
= username
Username with DB access

password
= secrets.MS_SQL
Password for username e.g. secrets.TOKEN, Secrets can be used to reference encrypted authentication credentials in Connectors. See the Secrets documentation for more details.

Input Mapping

databaseName
= databaseName
Name of the database containing the table

tableName
= tableName
Name of the table in which data needs to be inserted

dataToInsert
```
= [
    {
        "MEMBER_ID": 1,
        "member_name": "Alex",
        "Sys_date": "20230618 10:34:09 AM
    },
    {
        "MEMBER_ID": 2,
        "member_name": "Bob",
        "Sys_date": "2023-02-21T18:10:00"
    },
    {
        "MEMBER_ID": 3
    }
]
```
[{"columnName1": row1Col1Value, "columnName2": row1Col2Value...}, {"columnName1", row2Col1Value, "columnName2", row2Col2Value...}...] Data to insert into the table

Output Mapping

Result Variable
insertDataStatus
Name of variable to store the response in

Result Expression
=
Expression to map the response into process variables

Error Handling

Error Expression
=
Expression to handle errors. Details in the documentation.

To insert data into the table, take the following steps:

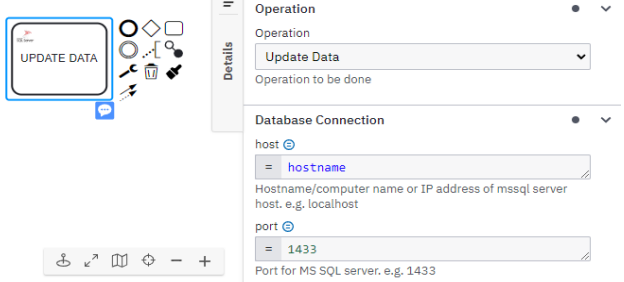
1. In the **Operation** section, set the field value **Operation** as **Insert Data**.
2. Set the required parameters and credentials in the **Database Connection** section. See the relevant appendix entry to find out more.
3. In the **Input Mapping** section, set the field **databaseName, tableName**.
4. Set **dataToInsert**, using FEEL expression as List of columns details, which is a List of context having keys as name, datatype and constraint.
5. We are following Insert syntax - **INSERT INTO tableName (columnNames) VALUES (*)** where columnNames is list of comma-separated column names extracted from keyset of first item in the dataToInsert List.

Insert Data operation response

You can use an output mapping to map the response:

2. Use **Result Variable** to store the response in a process variable. For example, `insertDataStatus`.

Update table Data


UPDATE DATA
Details
MS SQL DATABASE CONNECTOR
UPDATE DATA
General
Template
Applied
Operation
Operation
Update Data
Operation to be done
Database Connection
host
= hostname
Hostname/computer name or IP address of mssql server host. e.g. localhost
port
= 1433
Port for MS SQL server. e.g. 1433
username
= username
Username with DB access
password
= secrets.MS_SQL
Password for username e.g. secrets.TOKEN, Secrets can be used to reference encrypted authentication credentials in Connectors. See the Secrets documentation for more details.
Input Mapping
databaseName
= databaseName
Name of the database containing the table
tableName
= tableName
Table Name in which data needs to be updated
updateMap
= { "MEMBER_ID": "4", "member_name": "Delta" }
Context with column name and value to update data in table. e.g. {"columnName1":colValue, "columnName2":"colVal"}
filters
= { "filter": { "colName": "MEMBER_ID", "operator": "=", "value": 3 } }
Data Type: Map.of(String, Object), Desc: Filter Map can have 3 keys filter, logicalOperator & filterList. Simple filter e.g. {"filter": {"colName": "ColumnName", "operator": "=", "value": ColumnValue}}. If filter is null or empty, all rows in table will be updated
orderBy
= [{ "sortOn": "MEMBER_ID", "order": "descending" }]
Data Type: List.of(Map.of(String, String)), Desc: List of Maps will have 2 keys: sortOn and order. sortOn value will be column name for orderBy and order as ascending/descending. e.g. [{"sortOn":"ColumnName", "order":"ascending"}]
top
= 2
Data Type: Integer, Desc: Maximum number of rows to update
Output Mapping
Result Variable
updateDataStatus
Name of variable to store the response in
Result Expression
=
Expression to map the response into process variables
Error Handling
Error Expression
=
Expression to handle errors. Details in the documentation.

To update table data, take the following steps:

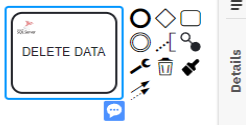
1. In the **Operation** section, set the field value **Operation** as **Update Data**.
2. Set the required parameters and credentials in the **Database Connection** section. See the relevant appendix entry to find out more.
3. In the **Input Mapping** section, set the field **databaseName, tableName**.
4. Set **updateMap**, using FEEL expression as context with key-value pairs for *columnName* & *value*. e.g. {"empAddress": "Krypton", "empName": "Kal-El"}
   These fields will update for all the rows which match the filter condition.
5. Set **filters**, using FEEL expression as context with keys as - filter, logicalOperator & filterList. e.g. {"filter":{"colName": "alias","operator": "like","value": "%superman%"}}
   These will used to construct the where clause for the SQL query.
6. Set **orderBy**, using FEEL expression as list of context with keys – sortOn and order. e.g. [{"sortOn": "powers","order": "descending"}]
   These will used to construct the orderBy clause for the SQL query. The order of rows to update.
7. Set **top**, the maximum number of rows to update.

Update Table Data operation response

You can use an output mapping to map the response:

1. Use **Result Variable** to store the response in a process variable. For example, `updateDataStatus`.

Delete table Data



Details

MS SQL DATABASE CONNECTOR
DELETE DATA

General

Template Applied

Operation

Operation

Delete Data

Operation to be done

Database Connection

host

= hostname

Hostname/computer name or IP address of mssql server host. e.g. localhost

port

= 1433

Port for MS SQL server. e.g. 1433

username

= username

Username with DB access

password

= secrets.MS_SQL

Password for username e.g. secrets.TOKEN, Secrets can be used to reference encrypted authentication credentials in Connectors. See the Secrets documentation for more details.

Input Mapping

databaseName

= databaseName

Name of the database containing the table

tableName

= tableName

Table/collection name to delete data from

filters

```
= {
    "filter": {
        "colName": "MEMBER_ID",
        "operator": ">=",
        "value": 1
    }
}
```

Data Type: Map.of(String, Object), Desc: Filter Map can have 3 keys filter, logicalOperator & filterList. Simple filter e.g. {"filter":{"colName": "ColumnName", "operator": "=", "value": ColumnValue}}

orderBy

```
= [
    {
        "sortOn": "MEMBER_ID",
        "order": "asc"
    }
]
```

Data Type: List.of(Map.of(String, String)), Desc: List of Maps will have 2 keys: sortOn and order. sortOn value will be column name for orderBy and order as ascending/descending. e.g. [{"sortOn":"ColumnName", "order":"ascending"}]

top

= 2

Maximum number of rows to delete

Output Mapping

Result Variable

deleteDataStatus

Name of variable to store the response in

Result Expression

=

Expression to map the response into process variables

Error Handling

Error Expression

=

Expression to handle errors. Details in the documentation.

To delete table data, take the following steps:

1. In the **Operation** section, set the field value **Operation** as **Delete Data**.
2. Set the required parameters and credentials in the **Database Connection** section. See the relevant appendix entry to find out more.
3. In the **Input** section, set the field **databaseName, tableName**.
4. Set **filters**, using FEEL expression as context with keys as - filter, logicalOperator & filterList. e.g. {"filter":{"colName": "alias","operator": "like","value": "%superman%"}} These will used to construct the where clause for the SQL query. All the matched rows will be deleted.
5. Set **orderBy**, using FEEL expression as list of context with keys – sortOn and order. e.g. [{"sortOn": "powers","order": "descending"}] These will used to construct the orderBy clause for the SQL query. The order of rows to delete.
6. Set **top**, the maximum number of rows to delete.

Delete Table Data operation response

You can use an output mapping to map the response:

1. Use **Result Variable** to store the response in a process variable. For example, `deleteDataOutput`.

Read table Data

MS SQL DATABASE CONNECTOR
READ DATA

Details

General

Template — Applied

Operation

Operation

Read Data

Operation to be done

Database Connection

host

= hostname

Hostname/computer name or IP address of mssql server host. e.g. localhost

port

= 1433

Port for MS SQL server. e.g. 1433

username

= username

Username with DB access

password

= secrets.MS_SQL

Password for username e.g. secrets.TOKEN, Secrets can be used to reference encrypted authentication credentials in Connectors. See the Secrets documentation for more details.

Input Mapping

databaseName

= databaseName

Name of the database containing the table

tableName

= tableName

Table/collection name to read data from

columnNames

= ["col1", "col2"]

List of columns/fields to return in result. If empty will return all columns/fields. e.g. ["columnName1", "columnName2",...]

filters

```
= {
    "filter": {
        "colName": "MEMBER_ID",
        "operator": "=",
        "value": 3
    }
}
```

Data Type: Map.of(String, Object), Desc: Filter Map can have 3 keys filter, logicalOperator & filterList. Simple filter e.g. {"filter": {"colName": "ColumnName", "operator": "=", "value": ColumnValue}}

orderBy

```
= [
    {
        "sortOn": "colName",
        "order": "descending"
    }
]
```

Data Type: List.of(Map.of(String, String)), Desc: List of Maps will have 2 keys: sortOn and order. sortOn value will be column name for orderBy and order as ascending/descending. e.g. [{"sortOn":"ColumnName", "order":"ascending"}]

top

= 100

Data Type: Integer, Desc: Maximum number of rows in output

Output Mapping

Result Variable

readDataStatus

Name of variable to store the response in

Result Expression

=

Expression to map the response into process variables

Error Handling

Error Expression

=

Expression to handle errors. Details in the documentation.

To read table data, take the following steps:

1. In the **Operation** section, set the field value **Operation** as **Read Data**.
2. Set the required parameters and credentials in the **Database Connection** section. See the relevant appendix entry to find out more.
3. In the **Input Mapping** section, set the field **databaseName, tableName**.
4. Set **columnNames**, using FEEL expression as List of columns to get in the output variable. e.g. [“col1”, “col2”]
5. Set **filters**, using FEEL expression as context with keys as - filter, logicalOperator & filterList. e.g. {"filter":{"colName": "alias","operator": "like","value": "%superman%"}} These will used to construct the where clause for the SQL query. All the matched rows will be returned in the output.
6. Set **orderBy**, using FEEL expression as list of context with keys – sortOn and order. e.g. [{"sortOn": "powers","order": "descending"}] These will used to construct the orderBy clause for the SQL query. The order of rows in output.
7. Set **top**, the maximum number of rows in output.

Read Table Data operation response

You can use an output mapping to map the response:

1. Use **Result Variable** to store the response in a process variable. For example, `readDataOutput`. It’s a List of Maps with keys as column name and value as respective row data.

Alter table

MS SQL DATABASE CONNECTOR
ALTER TABLE

Details

General

Template Applied

Operation

Operation

Alter Table

Operation to be done

Database Connection

host

= hostname

Hostname/computer name or IP address of mssql server host. e.g. localhost

port

= 1433

Port for MS SQL server. e.g. 1433

username

= username

Username with DB access

password

= secrets.MS_SQL

Password for username e.g. secrets.TOKEN, Secrets can be used to reference encrypted authentication credentials in Connectors. See the Secrets documentation for more details.

Input Mapping

databaseName

= databaseName

Name of the database containing the table

tableName

= tableName

Name of the table which needs to be altered

Method Type

Rename Table

Operation to be done

newTableName

= updatedTableName

New name of the table

Output Mapping

Result Variable

renameTableStatus

Name of variable to store the response in

Result Expression

=

Expression to map the response into process variables

Error Handling

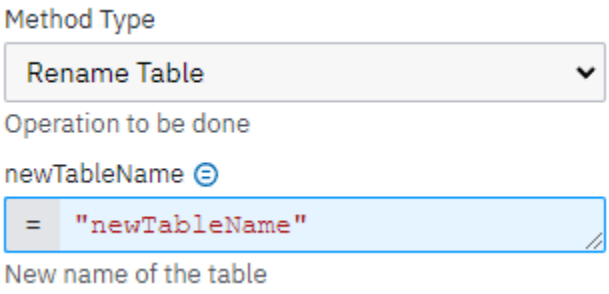
Error Expression

=

Expression to handle errors. Details in the documentation.

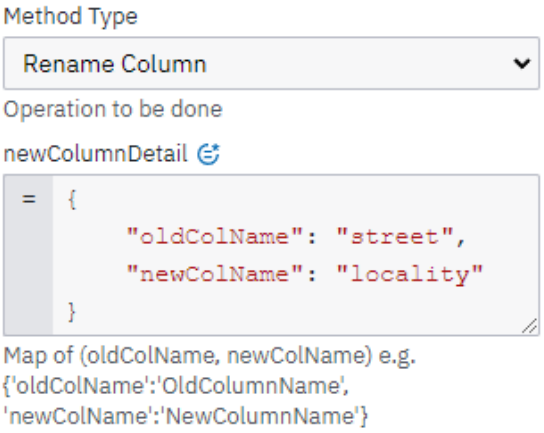
To alter table, take the following steps:

1. In the **Operation** section, set the field value **Operation** as **Alter Table**.
2. Set the required parameters and credentials in the **Database Connection** section. See the relevant appendix entry to find out more.
3. In the **Table Detail** section, set the field **databaseName, tableName**.
4. Set **Method Type**, types of alter operations –
    1. Rename Table



Rename table to **newTableName**

    2. Rename Column



Rename column (**oldColName**) to new name (**newColName**)

3. Add Constraint

Method Type

Add Constraint

Operation to be done

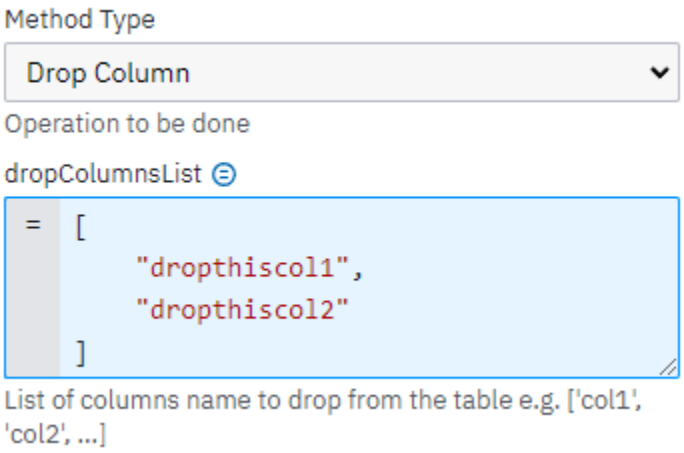
constraintDetails

```
= [
    {
        "Name": "UNIQUE",
        "Symbol": "uq_member_id",
        "Definition": "member_id"
    },
    {
        "Name": "Primary Key",
        "Symbol": "pk_member_id",
        "Definition": "member_id"
    },
    {
        "Name": "Foreign Key",
        "Symbol": "fk_member_id",
        "Definition": "(member_id) REFERENCES employee(empid)"
    },
    {
        "Name": "DEFAULT",
        "Symbol": "Sys_date",
        "Definition": "GETDATE() FOR Sys_date"
    },
    {
        "Name": "CHECK",
        "Symbol": "check_member_id",
        "Definition": "member_id>0"
    }
]
```

Details of constraints in the following format. e.g. [{'Name':'unique','Symbol':'constraint_symbol', 'Definition':'columnName'}, ...]

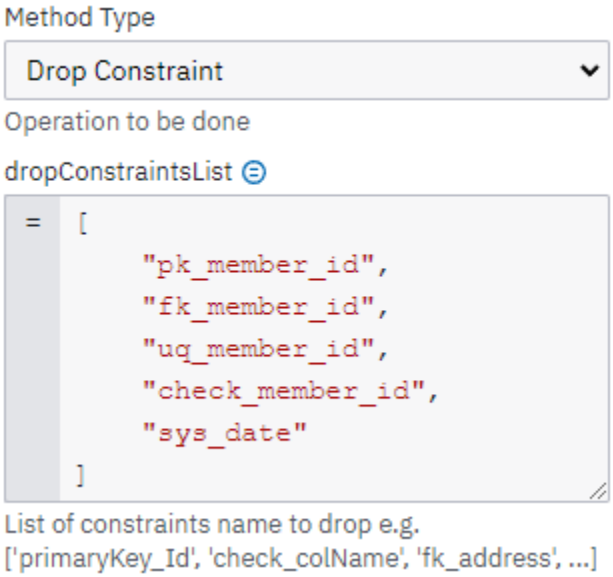
Add constraints to the table, use FEEL expression to provide input **constraintDetails** as List of contexts with keys as – name, symbol, and definition.
Name – Type of constraint e.g. UNIQUE, DEFAULT, PRIMARY KEY, FOREIGNKEY or CHECK
Symbol – The constraint name e.g. pk_id, fk_cin
Definition – Column name on which constraint needs to be applied

4. Drop Column



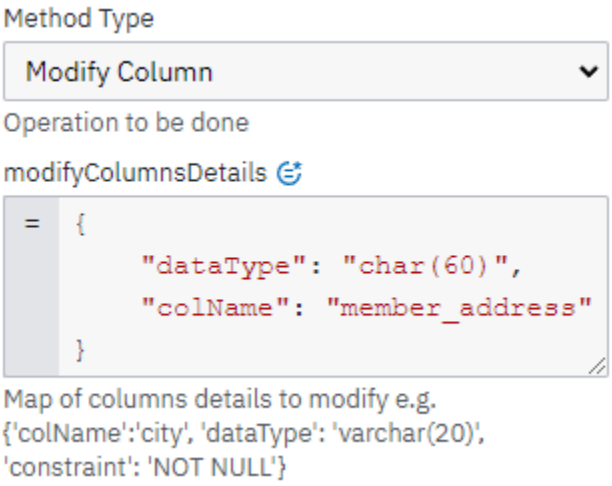
Drop column from the table, use FEEL expression to provide input **dropColumnsList** as List of columns.

5. Drop Constraint



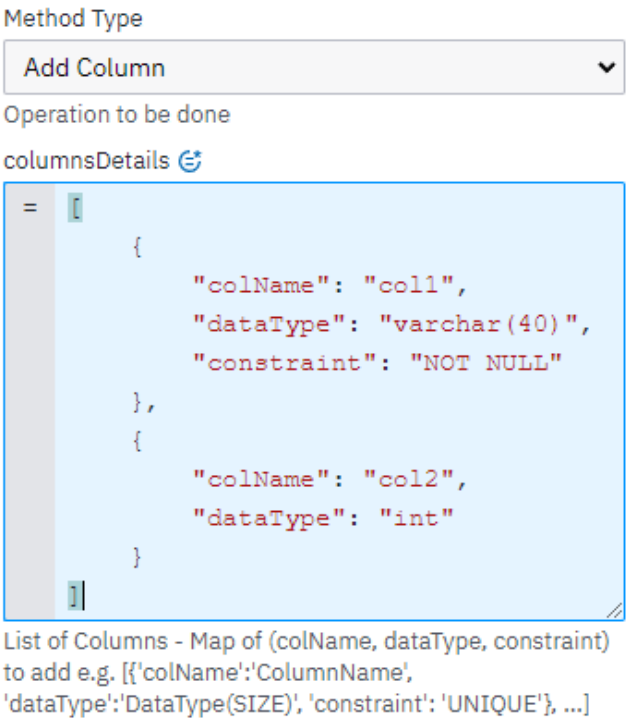
Drop constraint from the table, use FEEL expression to provide input **dropConstraintsList** as List of constraint name.

6. Modify Column



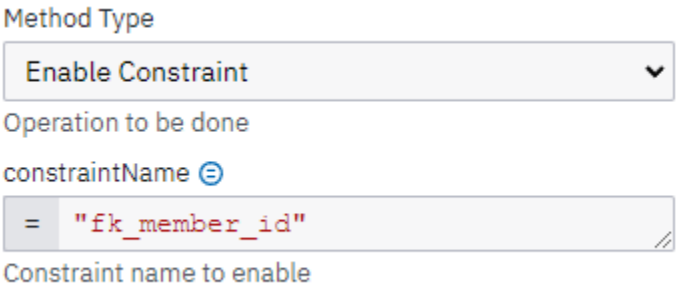
Set **modifyColumnsDetails**, using FEEL expression context, and it can have keys as – **colName**, **dataType** and **constraint**.
colName is mandatory and datatype or constraint can be provided to update.

7. Add Column

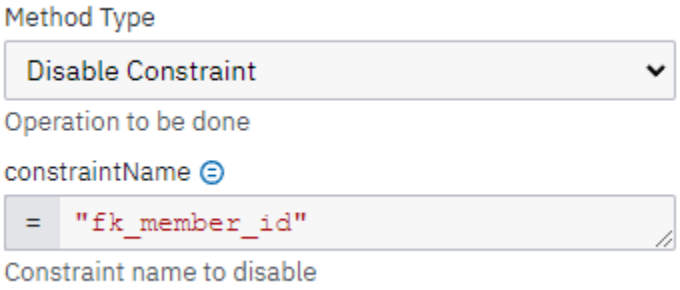


Set **columnsDetails**, using FEEL expression as List of contexts. Each context can have keys as **colName**, **dataType and constraint.**
name and dataType are mandatory.

8. **Enable Constraint**



9. Disable Constraint



Alter Table operation response

You can use an output mapping to map the response:

1. Use **Result Variable** to store the response in a process variable. For example, `alterTableOutput`.

Appendix & FAQ

Database Connection – Params values

Database connection group have 4 params – host, port, username, and password. These values will be used to connect to the database server.

How can I authenticate my Connector?

The **MSSQL Connector** needs the database credentials for connection. Hostname (host) – of the server where database is hosted, Port (port) – on which database server is running, Username (username) – User with proper privilege for operation and Password (password) – User password, which need to be saved as a Token in Secret vault and input can be provided as: secrets.TOKEN_NAME

What is filters input parameter?

Filters input is Map with keys – **filter**, **logicalOperator** and **filterList**.

1. `filter key's` value is a Map with keys – **colName**, **operator** and **value**.

colName – is column name to apply condition on.

Supported operators are –
[=, ==, equals, <>, not equals, <, less than, >, greater than, <=, less than or equals, >=, greater than or equals, like, in, is, not in, starts with, ends with]

value - is an Object and can be anything.

2. `logicalOperator key's` value can be OR, AND or NOT

3. `filterList key's` value is a list of Map with key filter. And value for this filter key must follow 1st point.

filter key can exist individually or with **optional logicalOperator** (value - NOT).
But **filterList and logicalOperator** both must be present, logicalOperator value will be used to club all filters in the filterList.

If **filterList** key is present in the main map, **filter** key will be **ignored**.

Internally it is being used for constructing **where clause** for SQL query.

Filters can be of two type –

1. Simple Filter – It will contain just one condition and may be a negation.
2. Complex Filter – It is collection of simple filters, clubbed using logical operator like AND/OR.

Examples:

Simple filter without negation

```
{"filter": {"colName": "alias", "operator": "like", "value": "%superman%"} }
```

Simple filter with negation

```
{"filter": {"colName": "alias", "operator": "like", "value": "%superman%"},
"logicalOperator": "NOT"}
```

Complex filter

```
{"logicalOperator": "AND", "filterList": [
        {"filter": {"colName": "empAddress","operator": "=","value": "Krypton"}},
        {"filter": {"colName": "empName","operator": "like","value": "%superman%"}},
        {"filter": {"colName": "age","operator": ">","value": 28}}
    ]
}
```

What is orderBy input parameter?

orderBy input is a List of Map with keys – sortOn and order. As the name suggests, internally it is being used to construct **order by clause**.

sortOn – contains the column name
order – a/asc/ascending **OR** d/desc/descending

```
[
    {
        "sortOn": "empId",
        "order": "descending"
    }
]
```

What is top input parameter?

top is the maximum number of rows for operation.
In case of read data it's maximum number of rows to return in the output.