

# Oracle Connector

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

## Prerequisites

To start working with the **Oracle 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 Oracle Connector task

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

To use an **Oracle 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.

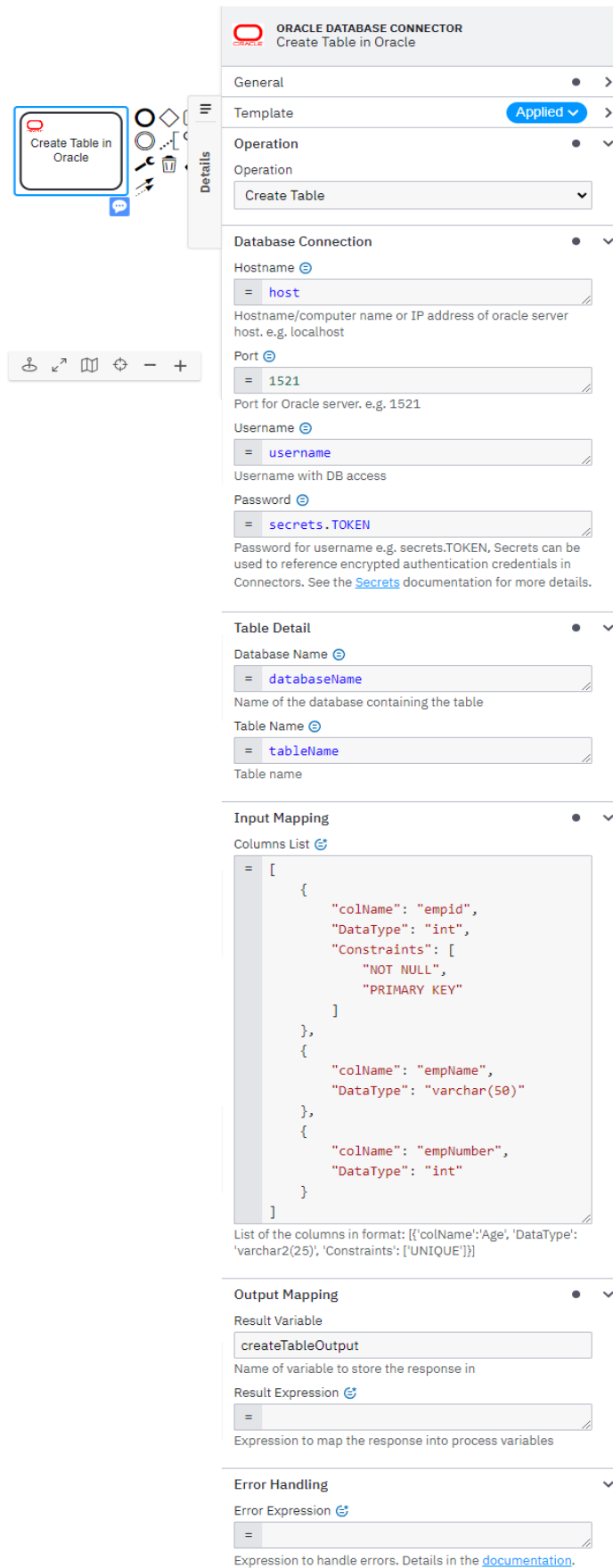
## Make your Oracle Connector executable

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

## Database connection Object input for Oracle Connector

**Oracle Connector** database connection object takes – host, port, username and password.  
e.g. localhost, 1521, username, password (as secrets Token e.g. secrets.ORACLE\_TOKEN )

## Create a new table



**ORACLE DATABASE CONNECTOR**  
Create Table in Oracle

General

Template **Applied**

Operation

Operation  
Create Table

Database Connection

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

Port  
= 1521  
Port for Oracle server. e.g. 1521

Username  
= username  
Username with DB access

Password  
= secrets.TOKEN  
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.

Table Detail

Database Name  
= databaseName  
Name of the database containing the table

Table Name  
= tableName  
Table name

Input Mapping

Columns List

```
= [
  {
    "colName": "empid",
    "DataType": "int",
    "Constraints": [
      "NOT NULL",
      "PRIMARY KEY"
    ]
  },
  {
    "colName": "empName",
    "DataType": "varchar(50)"
  },
  {
    "colName": "empNumber",
    "DataType": "int"
  }
]
```

List of the columns in format: [{colName:'Age', 'DataType': 'varchar2(25)', 'Constraints': ['UNIQUE']}]

Output Mapping

Result Variable  
createTableOutput  
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:

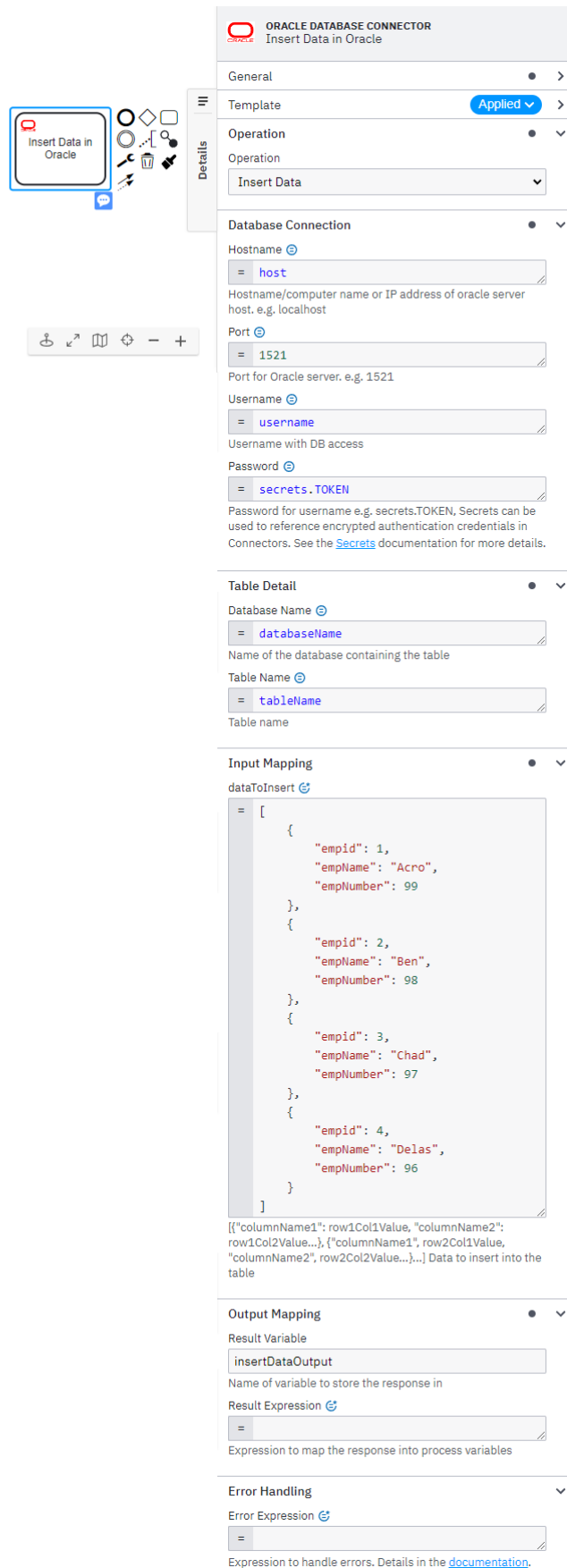
1. In the **Operation** section, set the field value **Operation** as **Create Table**.
2. Set the required parameters and credentials in the **Database Connection** section. See the relevant appendix entry to find out more.
3. In the **Table Detail** section, set the field **databaseName**, **tableName** 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 table data



**ORACLE DATABASE CONNECTOR**  
Insert Data in Oracle

General

Template **Applied**

Operation

Operation  
Insert Data

Database Connection

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

Port  
= 1521  
Port for Oracle server. e.g. 1521

Username  
= username  
Username with DB access

Password  
= secrets.TOKEN  
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.

Table Detail

Database Name  
= databaseName  
Name of the database containing the table

Table Name  
= tableName  
Table name

Input Mapping

dataToInsert

```
[
  {
    "empid": 1,
    "empName": "Acro",
    "empNumber": 99
  },
  {
    "empid": 2,
    "empName": "Ben",
    "empNumber": 98
  },
  {
    "empid": 3,
    "empName": "Chad",
    "empNumber": 97
  },
  {
    "empid": 4,
    "empName": "Delas",
    "empNumber": 96
  }
]
```

[{"columnName1": row1Col1Value, "columnName2": row1Col2Value...}, {"columnName1": row2Col1Value, "columnName2": row2Col2Value...}] Data to insert into the table

Output Mapping

Result Variable  
insertDataOutput  
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 table data, 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 **Table Detail** 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 ( columnName ) VALUES (\*)** where columnName 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 in Oracle

Details

**ORACLE DATABASE CONNECTOR**  
 Update Data in Oracle

General
Template
Operation

Template Applied

Operation
 

Update Data

Database Connection
 

Hostname 🔗  
 = host  
Hostname/computer name or IP address of oracle server  
host, e.g. localhost

Port 🔗  
 = 1521  
Port for Oracle server, e.g. 1521

Username 🔗  
 = username  
Username with DB access

Password 🔗  
 = secrets.TOKEN  
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.

Table Detail
 

Database Name 🔗  
 = databaseName  
Name of the database containing the table

Table Name 🔗  
 = tableName  
Table name

Input Mapping
 

updateMap 🔗  
 = {  
   "empid": 10,  
   "empName": "Zen",  
   "empNumber": 99  
 }  
Context with column name and value to update data in table.  
e.g. {"columnName1": colValue, "columnName2": colVal}

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

orderBy 🔗  
 = [  
   {  
     "sortOn": "empid",  
     "order": "desc"  
   }  
 ]  
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"}]

limit 🔗  
 = 1  
Data Type: Integer, Desc: Maximum number of rows to  
update

Output Mapping
 

Result Variable  
 updateMapOutput  
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 **Table Detail** 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 be 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 be used to construct the orderBy clause for the SQL query. The order of rows to update.
7. Set **limit**, 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**

Delete Data From Oracle Table

 **ORACLE DATABASE CONNECTOR**  
Delete Data From Oracle Table

General
>

Template
Applied >

Operation
>

Operation
>

Delete Data

Database Connection
>

Hostname
>

=

host

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

Port
>

=

1521

Port for Oracle server, e.g. 1521

Username
>

=

username

Username with DB access

Password
>

=

secrets.TOKEN

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.

Table Detail
>

Database Name
>

=

databaseName

Name of the database containing the table

Table Name
>

=

tableName

Table name

Input Mapping
>

filters
>

```

= {
  "filter": {
    "colName": "empid",
    "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
>

=

null

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"}]

limit
>

=

null

Maximum number of rows to delete

Output Mapping
>

Result Variable
>

deleteDataOutput

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:**

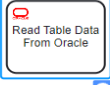
1. In the **Operation** section, set the field value **Operation** as **Delete Data**.
2. Set the required parameters and credentials in the **Database Connection** section. See the relevant appendix entry to find out more.
3. In the **Table Detail** section, set the field **databaseName**, **tableName**.
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 be 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 be used to construct the orderBy clause for the SQL query. The order of rows to delete.
6. Set **limit**, the maximum number of rows to delete.

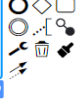

**Delete Table Data operation response**

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**



Details

Read Table Data From Oracle
ORACLE DATABASE CONNECTOR

Read Table Data From Oracle

Properties
Comments

General
>

Template
Applied >

Operation
>

Operation
>

Read Data
>

Database Connection
>

Table Detail
>

Input Mapping
>

Output Mapping
>

Error Handling
>

Hostname
>

=
host

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

Port
>

=
1521

Port for Oracle server, e.g. 1521

Username
>

=
username

Username with DB access

Password
>

=
secrets.TOKEN

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.

Database Name
>

=
databaseName

Name of the database containing the table

Table Name
>

=
tableName

Table name

columnNames
>

=
[]

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

filters
>

=

```
{
  "filter": {
    "colName": "name",
    "operator": "startsWith",
    "value": "A1"
  }
}
```

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": "empid",
    "order": "desc"
  }
]
```

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"}]

limit
>

=
0

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

Result Variable
>

=
tableData

Name of variable to store the response in

Result Expression
>

=

Expression to map the response into process variables

Error Expression
>

=

Expression to handle errors. Details in the [documentation](#).

**To read table data, take the following steps:**

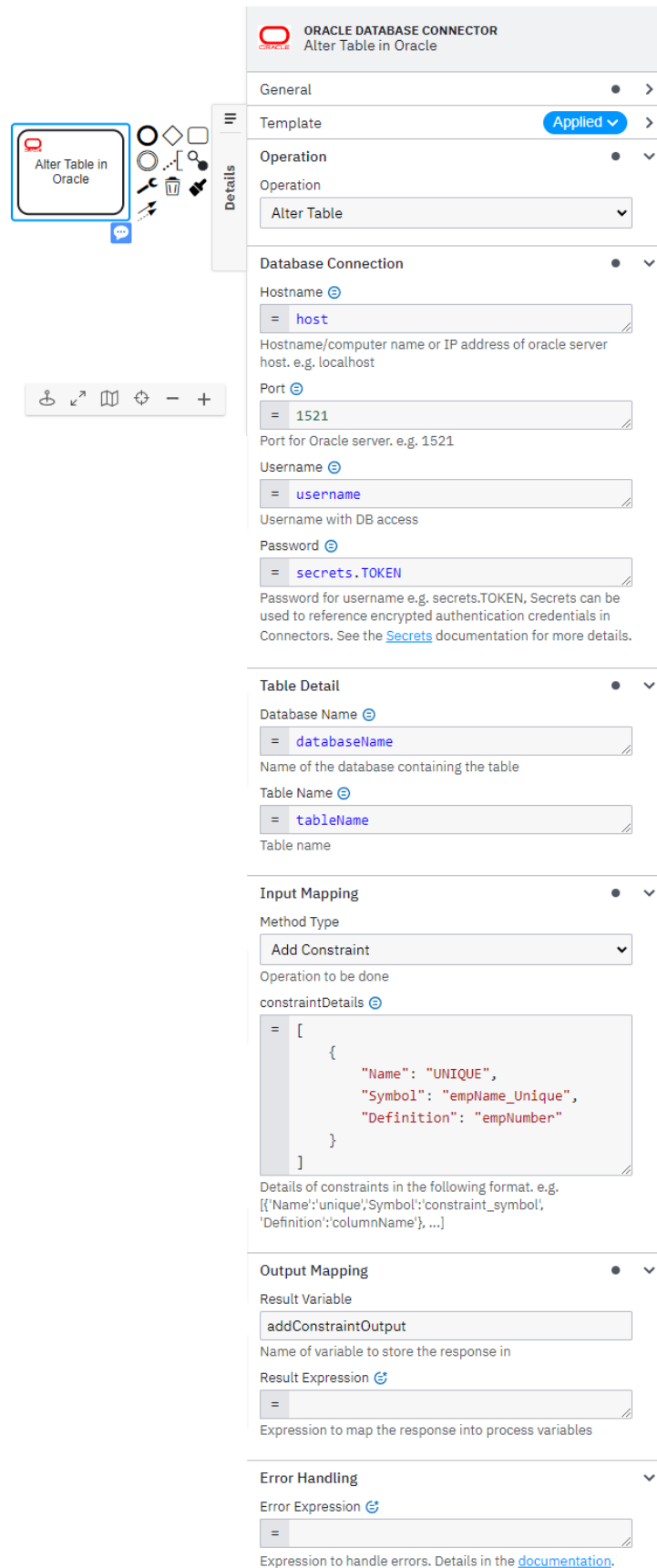
1. In the **Operation** section, set the field value **Operation** as **Read Data**.
2. Set the required parameters and credentials in the **Database Connection** section. See the relevant appendix entry to find out more.
3. In the **Table Detail** section, set the field **databaseName**, **tableName**.
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 be 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 be used to construct the orderBy clause for the SQL query. The order of rows in output.
7. Set **limit**, 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**



**ORACLE DATABASE CONNECTOR**  
Alter Table in Oracle

General

Template **Applied**

Operation

Operation

Alter Table

Database Connection

Hostname

= host

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

Port

= 1521

Port for Oracle server, e.g. 1521

Username

= username

Username with DB access

Password

= secrets.TOKEN

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.

Table Detail

Database Name

= databaseName

Name of the database containing the table

Table Name

= tableName

Table name

Input Mapping

Method Type

Add Constraint

Operation to be done

constraintDetails

= [

```
{
  "Name": "UNIQUE",
  "Symbol": "empName_Unique",
  "Definition": "empNumber"
}
```

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

Output Mapping

Result Variable

addConstraintOutput

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

Method Type

Rename Table ▼

Operation to be done

newTableName ⓘ

= "newTableName"

New name of the table

Rename table to **newTableName**

2. Rename Column

Method Type

Rename Column ▼

Operation to be done

newColumnDetail ⓘ

= {  
 "oldColName": "street",  
 "newColName": "locality"  
}

Map of (oldColName, newColName) e.g.  
{'oldColName':'OldColumnName',  
'newColName':'NewColumnName'}

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

3. Add Constraint

Method Type

Add Constraint

Operation to be done

constraintDetails ⓘ

```
= [
  {
    "name": "Unique",
    "symbol": "UC_EmpNumber",
    "Definition": "empNumber"
  },
  {
    "Name": "Primary Key",
    "Symbol": "PK_EmployeeID",
    "Definition": "empid"
  },
  {
    "Name": "Foreign Key",
    "Symbol": "FK_empID",
    "Definition": "(person_id) REFERE
  },
  {
    "name": "CHECK",
    "symbol": "ch_id",
    "Definition": "empid>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, 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

Method Type

Drop Column

Operation to be done

dropColumnsList ⓘ

```
= [
    "dropthiscol1",
    "dropthiscol2"
]
```

List of columns name to drop from the table e.g. ['col1', 'col2', ...]

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

## 5. Drop Constraint

Method Type

Drop Constraint

Operation to be done

dropConstraintsList ⓘ

```
= [
    "primaryKey",
    "empName_Unique"
]
```

List of constraints name or primaryKey e.g. ['primaryKey', 'check\_colName', 'fk\_address', ...]

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

## 6. Modify Column

Method Type

Modify Column

Operation to be done

modifyColumnsDetails [🔗](#)

```
= [
  {
    "colName": "address",
    "dataType": "varchar2(5)",
    "constraint": "NOT NULL"
  },
  {
    "colName": "empNumber",
    "dataType": "int"
  }
]
```

List of columns details to modify e.g.  
 [{ 'colName': 'city', 'dataType': 'varchar(20)',  
 'constraint': 'NOT NULL'}, ...]

Set **modifyColumnsDetails**, using FEEL expression as List of contexts. Each context can have keys as – **colName**, **dataType** and **constraint**.

**colName** is mandatory and datatype or constraint can be provided to update.

## 7. Add Column

Method Type

Add Column

Operation to be done

columnsDetails [🔗](#)

```
= [
  {
    "colName": "col1",
    "dataType": "varchar(40)",
    "constraint": "NOT NULL"
  },
  {
    "colName": "col2",
    "dataType": "int"
  }
]
```

List of Columns - Map of (colName, dataType, constraint)  
 to add e.g. [{ 'colName': 'ColumnName',  
 'dataType': 'DataType(SIZE)', 'constraint': 'UNIQUE'}, ...]

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

Method Type

Enable Constraint ▼

Operation to be done

constraintName ⓘ

= constraintNameToEnable

Constraint name to enable

## 9. Disable Constraint

Method Type

Disable Constraint ▼

Operation to be done

constraintName ⓘ

= constraintNameToDisable

Constraint name to disable

## 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 **Oracle 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?

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 1<sup>st</sup> 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?

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 limit input parameter?

Limit is the maximum number of rows for operation.

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