

FME Database Mastery



Take control of your
data with FME and
Databases



```
if _operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
elif _operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif _operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

#selection at the end -add back the deselected
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active = modifier_ob
print("Selected" + str(modifier_ob)) # modifier ob
#mirror_ob.select = 0
#one = bpy.context.selected_objects[0]
#bpy.data.objects[one.name].select = 1
except:
    print("please select exactly two objects, the

#----- OPERATOR CLASSES -----
# Mirror Tool

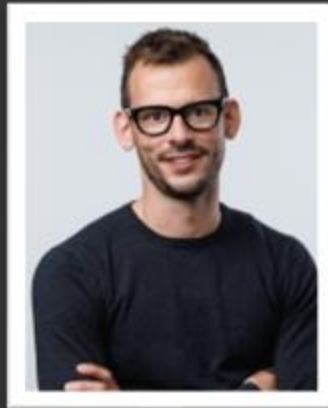
class MirrorX(bpy.types.Operator):
    """this adds an X mirror to the selected object"""
    bl_name = "Mirror X"
    bl_idname = "mirror_x"
    bl_options = {'REGISTER', 'UNDO'}

    @classmethod
    def poll(cls, context):
        obj = context.active_object
        if obj is None:
            return False
        if obj.type != 'MESH':
            return False
        if obj.mode != 'EDIT':
            return False
        return True

    def execute(self, context):
        obj = context.active_object
        mirror_mod = obj.modifiers.new('Mirror X', 'MIRROR')
        mirror_mod.use_x = True
        mirror_mod.use_y = False
        mirror_mod.use_z = False
        return {'FINISHED'}
```

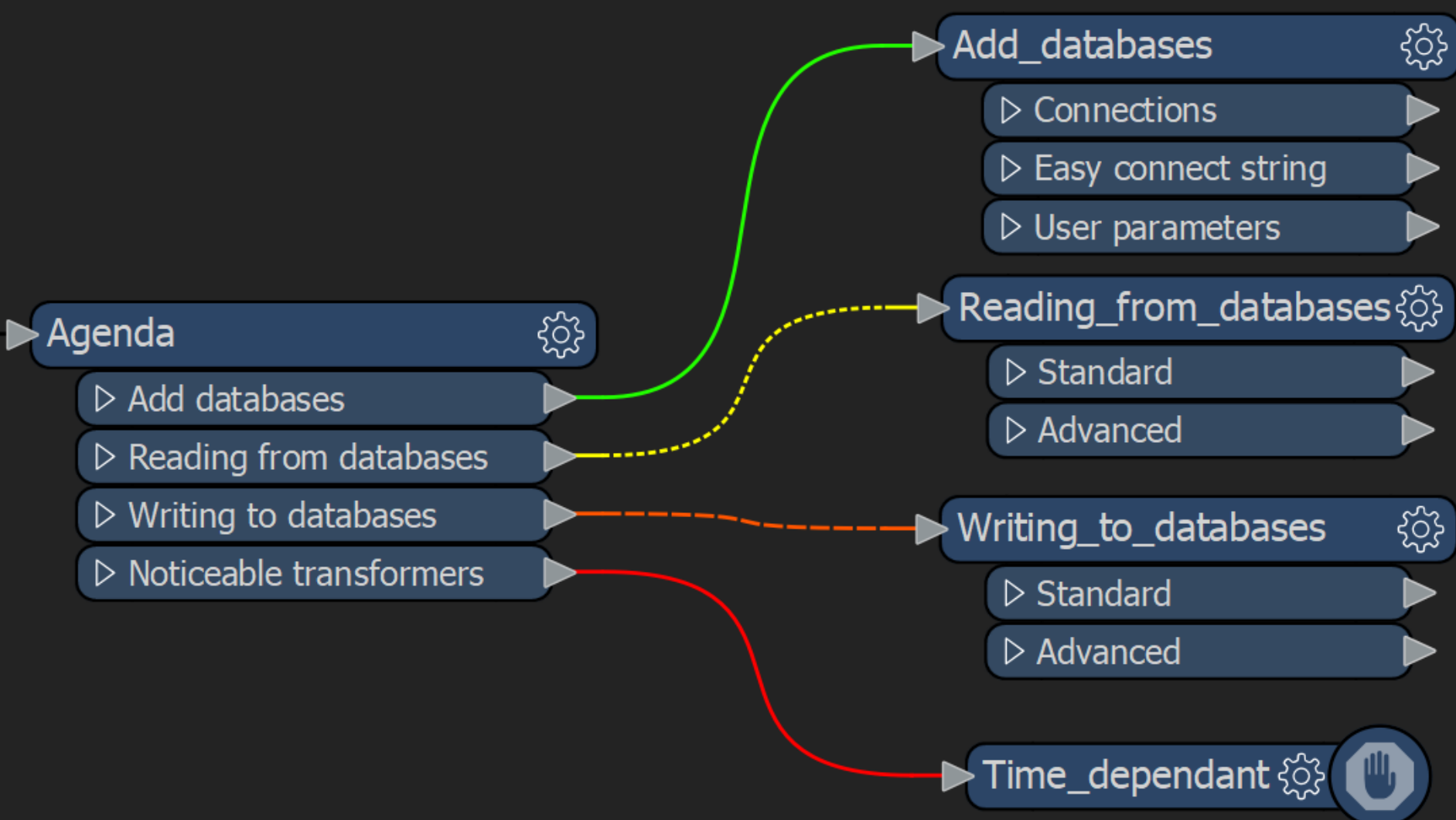
QUESTIONS

Jurgen
van Tiggelen



Mikael
Månsson







POLL x2

Add Databases



Connections
Easy connect string
User parameter

Edit 'train safe'

Name:

train safe Parameters

Host:

Port:

Database:

Username:

Password:

SSL Mode:

Oracle Non-Spatial Parameters

Database Connection

Service Name or Easy Connect:

Username:

Password:

Oracle Workspace: ...

Persistent Connection:

Include System Tables:



DEMO

Reading from databases

Pro's

- Easy and clear to use.

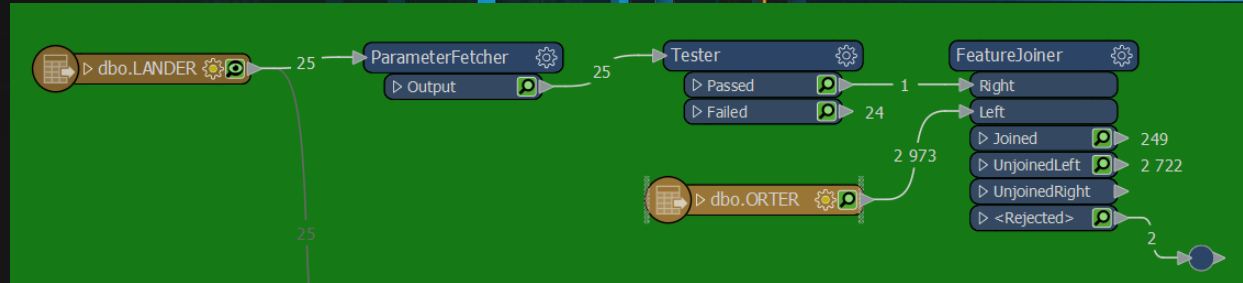
Con's

- Slower performances.
- Memory intensive.

```
if _operation == "MIRROR_X":  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
elif _operation == "MIRROR_Y":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
elif _operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True  
  
#selection at the end -add back the deselected  
mirror_ob.select= 1  
modifier_ob.select=1  
bpy.context.scene.objects.active = modifier_ob  
print("Selected" + str(modifier_ob)) # modifier ob  
#mirror_ob.select = 0  
#one = bpy.context.selected_objects[0]  
#bpy.data.objects[one.name].select = 1  
except:
```



Classic readers.
Combinations.
Where clauses.
SQL code.



Reading from databases

Pro's

- Easy and clear to use.
- Reading less data.
- Better performance.

Con's

- More complicated

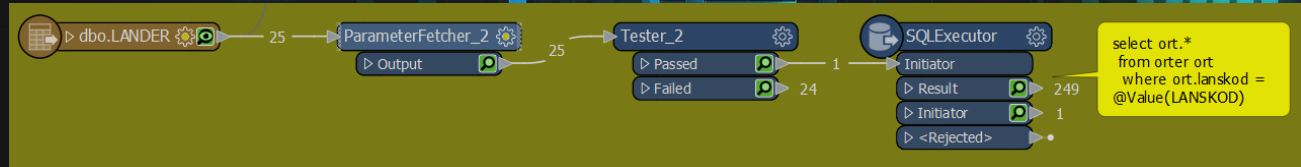
**This method can also be done with the DatabaseJoiner which doesn't require SQL code knowledge*

```
if _operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
elif _operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif _operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

#selection at the end -add back the deselected
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active = modifier_ob
print("Selected" + str(modifier_ob)) # modifier ob
#mirror_ob.select = 0
#one = bpy.context.selected_objects[0]
#bpy.data.objects[one.name].select = 1
except:
```



Classic readers.
Combinations.
Where clauses.
SQL code.



```
class MirrorX(bpy.types.Operator):
    """Mirror an object to the selected object"""
    bl_name = "MirrorX"
    bl_idname = "mirror_x"
    bl_options = {'REGISTER', 'UNDO'}

    @classmethod
    def poll(cls, context):
        return context.selected_objects > 0
```

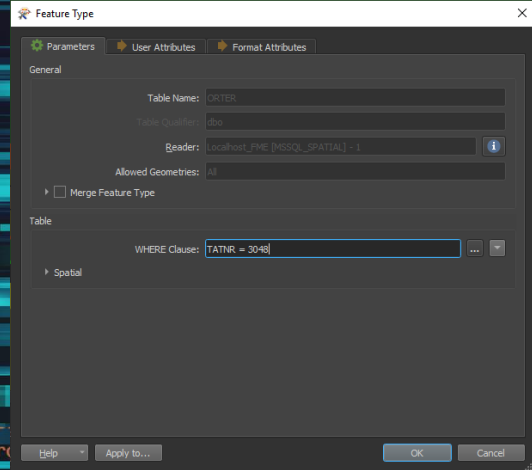
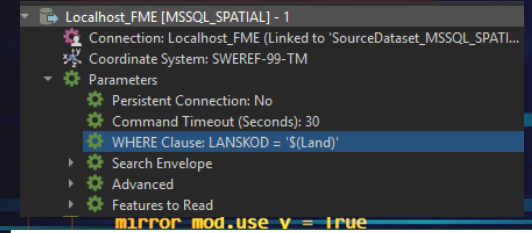

Reading from databases

Pro's

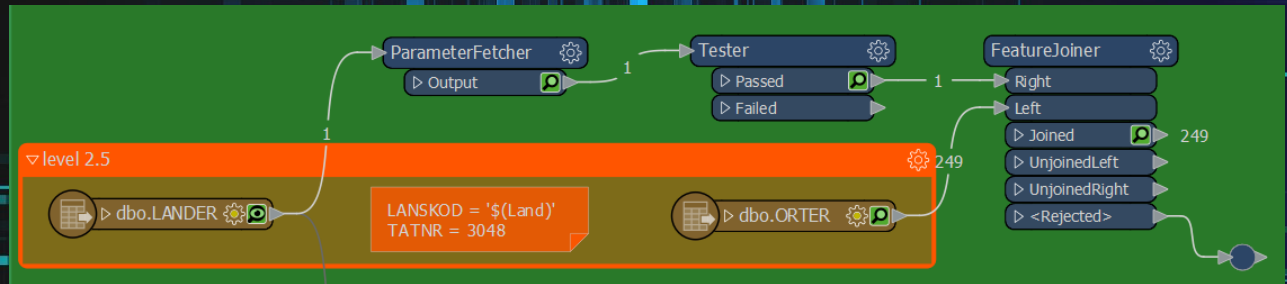
- Relative easy to use.
- Reading less data.
- Better performance.

Con's

- Where clauses are spread over tables and harder to see.



Classic readers.
Combinations.
Where clauses.
SQL code.



Reading from databases



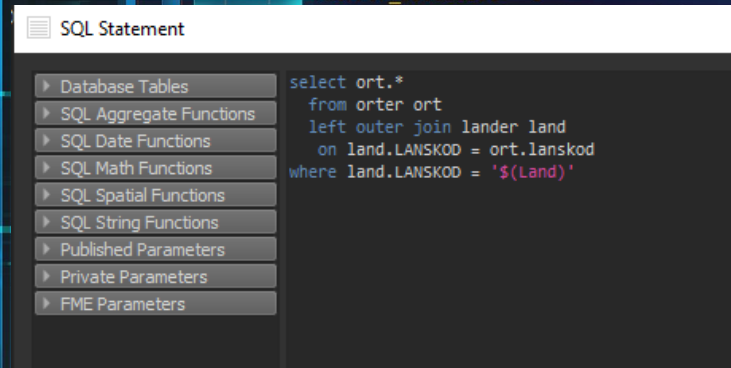
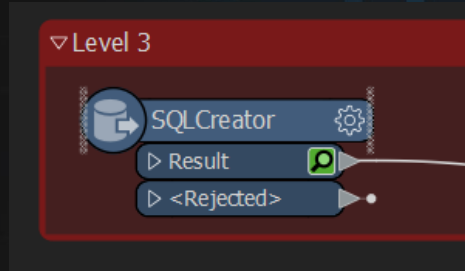
Classic readers.
Combinations.
Where clauses.
SQL code.

Pro's

- Best performance when query is well formed.
- Best memory usage.

Con's

- Most complicated.
- Requires SQL knowledge.



```
if _operation == "MIRROR_X":  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
elif _operation == "MIRROR_Y":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
elif _operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True  
  
#selection at the end -add back the deselected  
mirror_ob.select= 1  
modifier_ob.select=1  
bpy.context.scene.objects.active = modifier_ob  
print("Selected" + str(modifier_ob)) # modifier ob  
#mirror_ob.select = 0
```

Writing to databases



Classic writers.
Insert Statements.
Complicated statements.

The screenshot shows the 'Feature Type' dialog box in a GIS application. The 'Table' tab is active, and the 'Table Handling' dropdown is set to 'Create If Needed', which is highlighted with a red box. Other settings include 'Table Name: pop_density', 'Writer: Localhost_FME [MSSQL_SPATIAL] - 2', and 'Feature Operation: Insert'. The 'Spatial' section shows 'Spatial Type: Inherit from Writer'. The background shows a tree view with 'pop_density' selected.

- Always look at "Table Handling" before writing to a database.
- Use the Import featurtype from dataset function if the table already exists in the database to ensure correct datatypes.

Writing to databases



Classic writers.
Insert Statements.
Complicated statements.

This type of insert is by far harder than a classic writer but it allows you to call functions and sub-select statements for example.

The image shows two overlapping dialog boxes from the FME software. The top dialog is titled "SQLExecutor Parameters" and contains the following fields:

- Transformer Name: SQLExecutor_4
- Database: Format: Microsoft SQL Server Spatial
- Connection: Localhost_FME
- Coord. System: Read from source
- SQL Statement: [dbo].[pop_density]([Municipality], ...
- Attributes to Expose: (empty)
- Combine Attributes: Result Attributes Only
- Combine Geometry: Initiator Geometry Only

The bottom dialog is titled "SQL Statement" and shows a tree view on the left with "Database Tables" expanded to "FME Feature Attributes". The tree lists attributes such as _area, 0-17, 18-64, 65-, density, Divorced, LANSKOD, Married, Mean, Men, Municipality, Population, and Women. The main text area contains the following SQL code:

```
insert into [dbo].[pop_density]([Municipality],  
                                [Lanskod],  
                                [area],  
                                [Population],  
                                [density],  
                                [women],  
                                [men],  
                                [0-17],  
                                [18-64],  
                                [65- ],  
                                [Married],  
                                [Divorced],  
                                [Mean])  
VALUES ('@value(Municipality)',  
        (select ktkod from dbo.[LANDER] where [LANSKOD] = @value(LANSKOD)),  
        '@value(_area)',  
        '@value(Population)',  
        '@value(density)',  
        '@value(women)',  
        '@value(Men)',  
        '@value(0-17)',  
        '@value(18-64)',  
        '@value(65- )',  
        '@value(Married)',  
        '@value(Divorced)',  
        '@value(Mean)');
```

The SQLExecutor allows you to use more advanced sql statements.

```
if @operation = "MIRROR_X"  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
elif @operation == "MIRROR_Y":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True
```

Writing to databases



Classic writers.
Insert Statements.
Complicated statements.

The screenshot displays the FME software interface. On the left, a workflow diagram shows a 'SQLExecutor_5' tool connected to an 'Initiator' and a 'Result' tool. Below this, the 'SQLExecutor Parameters' dialog box is open, showing the following configuration:

- Transformer Name: SQLExecutor_5
- Database: Microsoft SQL Server Spatial
- Connection: Localhost_FME
- Coord. System: Read from source
- SQL Statement: `merge into [dbo].[pop_density] pop ...`
- Attributes to Expose: (empty)
- Combine Attributes: Result Attributes Only
- Combine Geometry: Initiator Geometry Only

On the right, the 'SQL Statement' editor is open, displaying a complex SQL script:

```
merge into [dbo].[pop_density] pop  
using (select '@value(Municipality)' as kommun pop2  
on (pop.[Municipality] = '@value(Municipality)')  
when not matched then  
insert([Municipality],  
[lanskod],  
[area],  
[Population],  
[density],  
[Women],  
[Men],  
[0-17],  
[18-64],  
[65- ],  
[Married],  
[Divorced],  
[Mean])  
values('@value(Municipality)',  
(select kkod from dbo.[LANDER] where [LANSKOD] = @value(LANSKOD)),  
'@value(_area)',  
'@value(Population)',  
'@value(density)',  
'@value(Women)',  
'@value(Men)',  
'@value(0-17)',  
'@value(18-64)',  
'@value(65- )',  
'@value(Married)',  
'@value(Divorced)',  
'@value(Mean)')  
when matched then  
update set [lanskod] = 222;
```

The SQLExecutor allows you to use more advanced sql statements and even create tables, functions, etc.

**Make sure that you know what you are doing with these kind of scripts. Talk with your databasemanager!*

```
if _operation = "MIRROR_Y"  
  mirror_mod.use_x = True  
  mirror_mod.use_y = False  
  trigger_mod.use_x = False  
  trigger_mod.use_y = True  
elif _operation == "MIRROR_Y":  
  mirror_mod.use_x = False  
  mirror_mod.use_y = True
```

Writing to databases



Classic writers.
Insert Statements.
Complicated statements.

The screenshot shows the 'SQL Statement' editor window with a tree view on the left and a code editor on the right. The tree view includes categories like Database Tables, FME Features, SQL Aggregate Functions, SQL Date Functions, SQL Math Functions, SQL String Functions, Published Parameters, Private Parameters, FME Parameters, String Functions, Math Functions, and Date/Time Functions. The code editor contains the following SQL script:

```
declare  
  cursor c is  
  select s.referenced_name seq_name  
        ,s.name      trg_name  
        ,t.table_name  
        ,q.last_number  
  from user_dependencies s  
       ,user_triggers    t  
       ,user_sequences   q  
  where s.referenced_type = 'SEQUENCE' and s.name like '%M%' and q.sequence_name = s.referenced_name  
       and t.trigger_name = s.name and t.triggering_event = 'INSERT';  
  
  l_max_val number;  
  
  procedure sequence_newvalue(seqname varchar2  
                             ,newvalue number) as  
  in number;  
  ib number;  
  begin  
    select nvl(last_number  
            ,1)  
           ,nvl(increment_by  
            ,1)  
    into ln  
           ,ib  
    from user_sequences  
    where sequence_name = upper(seqname);  
  
    DBMS_OUTPUT.PUT_LINE ('newvalue = ' || newvalue );  
    DBMS_OUTPUT.PUT_LINE ('ln = ' || ln);  
    DBMS_OUTPUT.PUT_LINE ('ib = ' || ib);  
    execute immediate 'ALTER SEQUENCE ' || seqname || ' INCREMENT BY ' || (newvalue - ln);  
  
    execute immediate 'SELECT ' || seqname || ',NEXTVAL FROM DUAL'  
    into ln;  
  
    execute immediate 'ALTER SEQUENCE ' || seqname || ' INCREMENT BY ' || ib;  
  end;  
  
  for r in c  
  loop  
    execute immediate 'SELECT max(id) from ' || r.table_name  
    into l_max_val;  
  
    if l_max_val > r.last_number then  
      DBMS_OUTPUT.PUT_LINE(r.seq_name || ' (' || r.last_number || ') l_max_val = ' || l_max_val);  
      sequence_newvalue(seqname => r.seq_name  
                        ,newvalue => l_max_val+1);  
    end if;  
  end loop;  
end;
```

the deselected
modifier ob
) # modifier ob
[0]
= 1
objects, the



DEMO

Online Training

23-25/6 FME Desktop Basic Training

<https://dataflow.center/training-events/>

Upcoming webinars

17/6 Special guest: Dmitri Bach from Safe Software

Lets's go gaming with FME!

25/6 FME Server Apps

<https://dataflow.center/training-events/>

Contact



fmesupport@sweco.se

jurgen.van.tiggelen@sweco.se

mikael.mansson@sweco.se

<https://dataflow.center/contact>