```python
#Use chemdraw API to convert cdxml to sdf
# deficiency: chemdraw has be to closed and opened everytime. This is not the case for cdx 2 cdxml conversion
import glob
import comtypes.client as w32
import datetime



#assemble list of cdx or CDXML files

#file_list = glob.glob('D:/patent1/**/*.cdx', recursive = True)
file_list = glob.glob('D:\\Patent1\\Structures_CDXML_C1\\*.cdxml', recursive = True)



# Creates invisible ChemDraw object.
#ChemDraw = w32.CreateObject("ChemDraw.Application")

n = (len(file_list))
print(n)


current_time = datetime.datetime.now()  #check time
print (current_time) #to estimate time needed for the job using test run

i = 1
for file in file_list:
    # Creates invisible ChemDraw object.
    ChemDraw = w32.CreateObject("ChemDraw.Application")
    file_pointer = ChemDraw.Documents.Open(file)
    document = ChemDraw.Documents.Item(i)

    # Save document.
    location = 'D:\\Patent1\\Structures_CDXML_C1\\' + file[31:62] + '.sdf'
    print(location)
    document.SaveAs(location)

    # Close Document.
    document.Close()
    # Closes ChemDraw
    ChemDraw.Quit()

current_time = datetime.datetime.now() #check time
print (current_time)
```

```
#Create new table from parent table (Ironpython in spotfire)
import System
from Spotfire.Dxp.Data import DataType,DataTableSaveSettings
from Spotfire.Dxp.Data.Import import TextFileDataSource,TextDataReaderSettings
from System.IO import Path,StreamWriter,StringReader,StreamWriter,MemoryStream,SeekOrigin
from Spotfire.Dxp.Application.Visuals import VisualContent
from Spotfire.Dxp.Application.Visuals import CrossTablePlot
from Spotfire.Dxp.Data import DataValueCursor,RowSelection,IndexSet

def getVisual(visualTitle):
     for page in Document.Pages:
          for vis in page.Visuals:
               if vis.Title == visualTitle:
                    return vis.As[VisualContent]()

#Modify the line below to specify the title of the cross table visualization
vis = getVisual("My_Cross_Table")

stream=MemoryStream()
writer=StreamWriter(stream)
vis.ExportText(writer)

stream.Seek(0,SeekOrigin.Begin)

readerSettings=TextDataReaderSettings()
readerSettings.Separator="\t"
readerSettings.AddColumnNameRow(0)

textDataSource =TextFileDataSource(stream,readerSettings)

if Document.Data.Tables.Contains("DataFromCrossTable"):
     Document.Data.Tables["DataFromCrossTable"].ReplaceData(textDataSource)
else:
     newTable = Document.Data.Tables.Add("DataFromCrossTable", textDataSource)
     tableSettings = DataTableSaveSettings (newTable, False, False)
     Document.Data.SaveSettings.DataTableSettings.Add(tableSettings)

myTable = Document.Data.Tables["DataFromCrossTable"]

n = 0
for col in myTable.Columns:
     n = n + 1
     if n == 1:
          MyCol = col.Name

LastRow = myTable.RowCount + 1
rowsToRemove=IndexSet(myTable.RowCount,False)

# Reference to the Column of the Table
dataValuesCursor=DataValueCursor.CreateFormatted(myTable.Columns[MyCol])

i = 0
for row in myTable.GetRows(dataValuesCursor):
     i = i + 1
  #Statement to remove or keep rows.
if i == LastRow:
     rowsToRemove.AddIndex(row.Index)

myTable.RemoveRows(RowSelection(rowsToRemove))
    # change date type from string to real
from Spotfire.Dxp.Data.Transformations import ExpressionTransformation,ColumnSelection
from Spotfire.Dxp.Data import *

table = Document.Data.Tables['DataFromCrossTable']
rowsToInclude = IndexSet(table.RowCount,True)
t = ExpressionTransformation()
```

```
#get the list of columns
columns = table.Columns
for cc in columns:
    ccname=cc.Name
    #ccname='Avg(standard_value)'
    cursor = DataValueCursor.CreateFormatted(table.Columns[ccname])
    values=[]
    for row in table.GetRows(rowsToInclude,cursor):
        values.append(cursor.CurrentValue)
        try:
            values = [float(x) for x in values ]
            t.ColumnReplacements.Add(
            ccname,'real(['+ccname+'])',ColumnSelection(ccname))
            # print ('datatypechanged')
        except:
            t.ColumnReplacements.Add(
            ccname,'string(['+ccname+'])',ColumnSelection(ccname))
            # print (ccname,'original column did not contain real data')
            pass
table.AddTransformation(t)
```

```python
#combine mol files into sdf and add properties to each molecule, including the location of its tiff format
#the code is used to construct US patent strucsture database
from rdkit import Chem
from rdkit.Chem import AllChem
#glob to get all the mol files
import glob

#assemble list of mol files

#sdfFile = open("D:\patent_sdf_new\I2023.sdf") #open sdf file in append mode
file_list = glob.glob('H:/patent1/I2023*/**/*.mol',
                      recursive = True)



print(len(file_list))
#write mol files to SD

with Chem.SDWriter("D:\patent_sdf_new\I2023.sdf") as w: #does not have to be existing file
    for file in file_list:
        m = Chem.MolFromMolFile(file) # read mol to rdkit molecule
        if m is None: #some mol files can't be read into rdkit
#             print(file) #name of files that can't be recognized
            m = Chem.MolFromSmiles('C1=CC=CN=C1') #add random structure so compound can be recorded in sdf
            m.SetProp("name", file[46:77]) #add property
            location = 'D:' + file[2:78] + 'tif'
            m.SetProp("location", location) #add property
            folder= 'D:' + file[2:45]
            m.SetProp("folder", folder)   #add property
            try:
                w.write(m)
            except:
                print(file, "can't be kekulized")

        else:
#           print(file)
#           print(m)
            m.SetProp("name", file[46:77])
            location1 = 'D:' + file[2:78] + 'tif'
            m.SetProp("location", location1)
            folder1= 'D:' + file[2:45]
            m.SetProp("folder", folder1)
            try:
                w.write(m)
            except:
                print(file, "can't be kekulized")
#sdfFile.close()
w.close()


```

```
-- DB browser code for assemble different tables
DROP TABLE IF EXISTS aa;
CREATE TABLE aa AS SELECT

--group_concat(actp.type) AS type_g,
--group_concat(actp.relation) AS relation_g,
--group_concat(actp.value) AS value_g,
--group_concat(actp.units) AS units_g,
--group_concat(actp.text_value) AS text_value_g,
group_concat(actp.standard_type) AS standard_type_g,
group_concat(actp.standard_relation) AS standard_relation_g,
group_concat(actp.standard_value) AS standard_value_g,
group_concat(actp.standard_units) AS standard_units_g,
group_concat(actp.standard_text_value) AS standard_text_value_g,
--group_concat(actp.activity_id) AS activity_id_g,
--group_concat(actp.comments) AS comments_g,
--group_concat(actp.result_flag) AS result_flag_g,
substr(group_concat(DISTINCT di.max_phase_for_ind ORDER by di.max_phase_for_ind
DESC), 1,1) AS di_max_phase_for_ind_g,
actp.activity_id,
md.chembl_id,
md.molregno,
a.description as 'a_description',
cs.canonical_smiles,
td.pref_name as 'td_pref_name',
a.assay_organism as 'a_assay_organism',
a.assay_tissue as 'a_assay_tissue',
a.assay_type as 'a_assay_type',
a.assay_test_type as 'a_assay_test_type',

--actp.type as 'actp.type',
--actp.standard_type as 'actp.standard_type',
--actp.standard_relation,
--actp.standard_value,
--actp.standard_units,
--actp.standard_text_value,
--act.type as 'act.type',
act.pchembl_value,
act.standard_type as 'act_standard_type',
act.type as 'act_type',
act.standard_relation as 'act_standard_relation',
act.standard_value as 'act_standard_value',
act.standard_units as 'act_standard_units',
act.activity_id as 'act_activity_id',
act.record_id as 'act_record_id',
act.doc_id as 'act_doc_id',

md.first_approval,
md.oral
```

```
--ms.synonyms,
--group_concat(ms.synonyms) AS ms.synonyms_g,
--ms.syn_type,
--group_concat(ms.syn_type) AS ms.syn_type,
--group_concat(rc.company) AS rc.company_g,
--rc.company

--FROM (Select DISTINCT molregno, max_phase_for_ind from drug_indication di Where
di.max_phase_for_ind = (SELECT MAX(max_phase_for_ind) FROM drug_indication)) AS di
FROM (Select DISTINCT molregno, max_phase_for_ind from drug_indication di) AS di
--FROM (Select molregno, max_phase_for_ind from drug_indication di Where
di.max_phase_for_ind IS NOT NULL) AS di
--LEFT JOIN (Select* From activities act WHERE act.standard_value IS NOT NULL AND
act.molregno IN ('397524')) as act ON di.molregno = act.molregno --select specific
compounds
LEFT JOIN (Select* From activities act WHERE act.standard_value IS NOT NULL) as act
ON di.molregno = act.molregno
LEFT JOIN molecule_dictionary md ON act.molregno = md.molregno --AND md.molregno =
'397524'
LEFT JOIN assays a ON act.assay_id = a.assay_id
LEFT JOIN compound_structures cs ON md.molregno = cs.molregno
LEFT JOIN target_dictionary td ON a.tid = td.tid
LEFT JOIN activity_properties actp ON act.activity_id = actp.activity_id
--LEFT JOIN molecule_synonyms ms ON md.molregno = ms.molregno
--LEFT JOIN research_companies rc ON ms.res_stem_id = rc.res_stem_id
WHERE md.molecule_type = 'Small molecule' --AND md.molregno IN ('397524')
GROUP BY md.chembl_id, act.activity_id
ORDER BY act.molregno DESC, act.activity_id DESC
LIMIT 2000000;
```