Inhaltsverzeichnis

Parserbau mit AntLR4

Falls man's noch mal braucht:

virtuelle Umgebnung installieren

```
python -m venv .venv
```

aktivieren

```
.venv\Scripts\Activate.bat
```

antlr4- jar (https://www.antlr.org/download/antlr-4.10.1-complete.jar) runterladen, im .venv Pfad speichern (.venv\scripts) und eine passende batch- datei dazu mit dem jar als classpath:

```
java -cp %~dp0antlr-4.10.1-complete.jar org.antlr.v4.Tool %*
```

dann die beiden *-g4 Files von https://github.com/antlr/grammars-v4/tree/master/sql/plsql laden und die dazugehörigen Basisklassen aus dem dortigen Python3- Ordner

die antlr4- Laufzeit installieren

```
pip install antlr4-python3-runtime
```

die *.g4 Scripte in Python umrechnen lassen

```
antlr4 -Dlanguage=Python3 *.g4 -o dist
```

und dann in einen Rumpf einbauen:

```
import sys
import os
import re
import json
from antlr4 import *
from pprint import pprint
from dist.PlSqlLexer import PlSqlLexer
from dist.PlSqlParser import PlSqlParser
from dist.PlSqlParserVisitor import PlSqlParserVisitor
from antlr4.error.ErrorListener import ErrorListener
from antlr4.error.ErrorStrategy import DefaultErrorStrategy
from antlr4.error.Errors import RecognitionException, NoViableAltException,
InputMismatchException,  FailedPredicateException,
ParseCancellationException


def eprint(*args, **kwargs):
    print(*args, file=sys.stderr, **kwargs)
```

```
class MyErrorListener( ErrorListener ):

    def __init__(self,file_name):
        self.file_name = file_name
        super(MyErrorListener, self).__init__()

    def handle_error(self):
        pass # eprint (f"Syntax error in {self.file_name}")

    def syntaxError(self, recognizer, offendingSymbol, line, column, msg,
e):
        self.handle_error()

    def reportAmbiguity(self, recognizer, dfa, startIndex, stopIndex, exact,
ambigAlts, configs):
        self.handle_error()

    def reportAttemptingFullContext(self, recognizer, dfa, startIndex,
stopIndex, conflictingAlts, configs):
        self.handle_error()

    def reportContextSensitivity(self, recognizer, dfa, startIndex,
stopIndex, prediction, configs):
        self.handle_error()

class MyErrorStrategy(DefaultErrorStrategy):

    def __init__(self,file_name):
        self.file_name = file_name
        super().__init__()



    def reportError(self, recognizer:Parser, e:RecognitionException):
        eprint (f"Syntax error 1 in {self.file_name}")

    def reportInputMismatch(self, recognizer:Parser,
e:InputMismatchException):
        eprint (f"Syntax error 2 in {self.file_name}")

    def reportNoViableAlternative(self, recognizer:Parser,
e:NoViableAltException):
        eprint (f"Syntax error 3 in {self.file_name}")

class MyVisitor(PlSqlParserVisitor):

    def __init__(self):
        self.results = {}
        super().__init__()
        self.file_name = ""
        self.regex_float = re.compile(r"\d+\.\d*", re.IGNORECASE)
```

```
        self.regex_int = re.compile(r"\d{3,}", re.IGNORECASE)

    def store_comand(self, cmd_string):
        line = self.regex_float.sub("<float>", cmd_string)
        line = self.regex_int.sub("<integer>", line)
        if not line in self.results:
            self.results[line] = set()
        self.results[line].add(self.file_name)

    def visitUnit_statement(self, ctx):
        value = ctx.getText()
        self.store_comand(value)
        return super().visitUnit_statement(ctx)

    def visitSql_plus_command(self, ctx):
        value = ctx.getText()
        self.store_comand(value)
        return super().visitSql_plus_command(ctx)

    def visitWhenever_command(self, ctx):
        value = ctx.getText()
        self.store_comand(value)
        return super().visitWhenever_command(ctx)

    def visitSql_script(self, ctx):
        value = ctx.getText()
        #print ("visitSql_script",value)
        return super().visitSql_script(ctx)

    """
     def visitSql_plus_command(self, ctx):
        value = ctx.getText()
        print (value)
        return super().visitSql_plus_command( ctx)


   def visitConstant(self, ctx):
        value = ctx.getText()
        print ("constant",value)
        return super().visitConstant( ctx)

    def visitNumeric(self, ctx):
        value = ctx.getText()
        print ("numeric",value)

        super().visitNumeric( ctx)
        return "bla"

    """
class SetEncoder(json.JSONEncoder):
    def default(self, obj):
```

```
        if isinstance(obj, set):
            return list(obj)
        return json.JSONEncoder.default(self, obj)


if __name__ == "__main__":
    #sqldir = "."
    sqldir = "mydir"
    visitor = MyVisitor()
    for file in os.listdir(sqldir):
        if file.endswith(".sql"):
            file_path = (os.path.join(sqldir, file))
            visitor.file_name = file
            # filter for backup files
            if "BAK" in file.upper():
                continue
            if "BACKUP" in file.upper():
                continue
            #print(file)
            with open(file_path) as fin:
                data = InputStream(fin.read().upper())
                # lexer
                lexer = PlSqlLexer(data)
                stream = CommonTokenStream(lexer)
                # parser
                parser = PlSqlParser(stream)
                parser.addErrorListener( MyErrorListener(file) )
                parser._errHandler = MyErrorStrategy(file)
                tree = parser.sql_script()
                # evaluator
                output = visitor.visit(tree)
                # print(output)
    for cmd, files in visitor.results.items():
        file_names=", ".join(files)
        print(f"{cmd}\t{len(files)}\t{file_names}")

    with open('parse_sql_data.json', 'w') as outfile:
        json.dump(visitor.results, outfile, cls=SetEncoder, indent=4)
```