

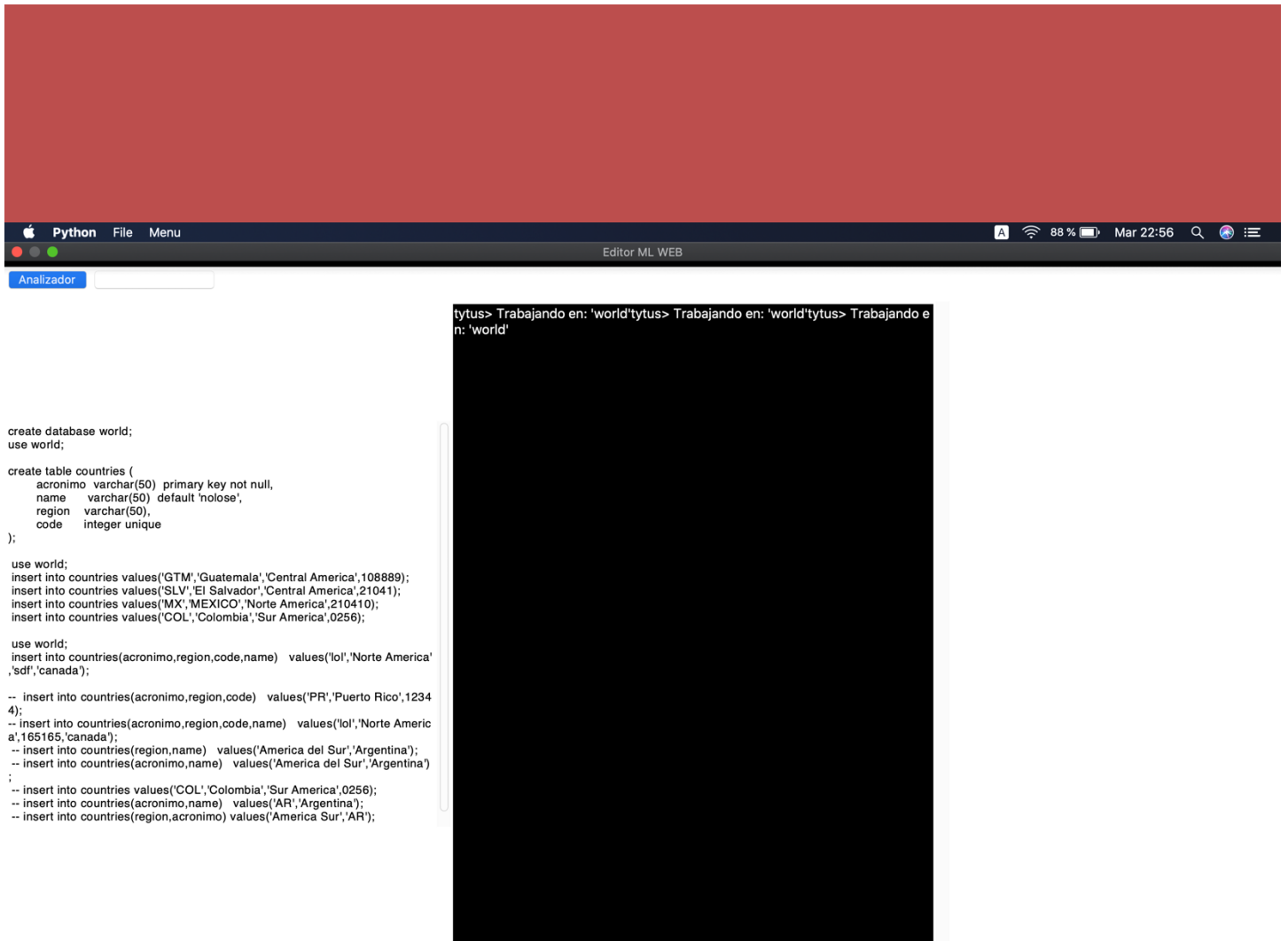
TytusDB

Manual Tecnico OLC2 fase 1

**Universidad San Carlos
de Guatemala**

Grupo 17

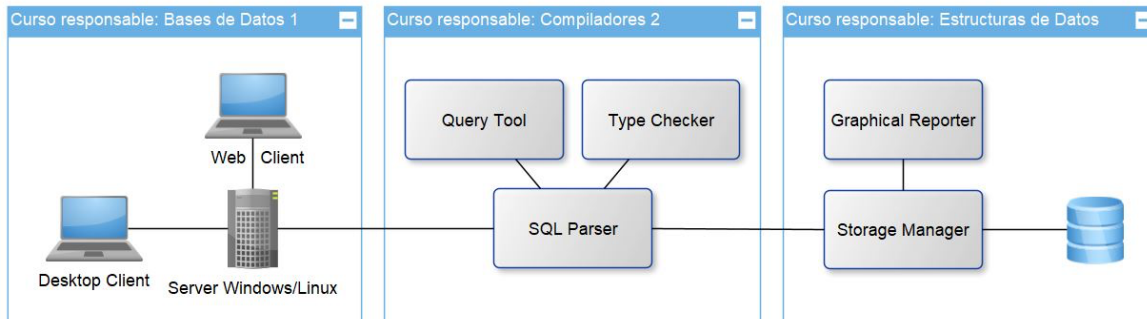
Pablo Rodrigo Barillas	201602503
Christopher Jhoanis Soto	201602569
Edgar Jonathan Arrecis	201602633
Nery Eduardo Herrera	201602870



Informacion General

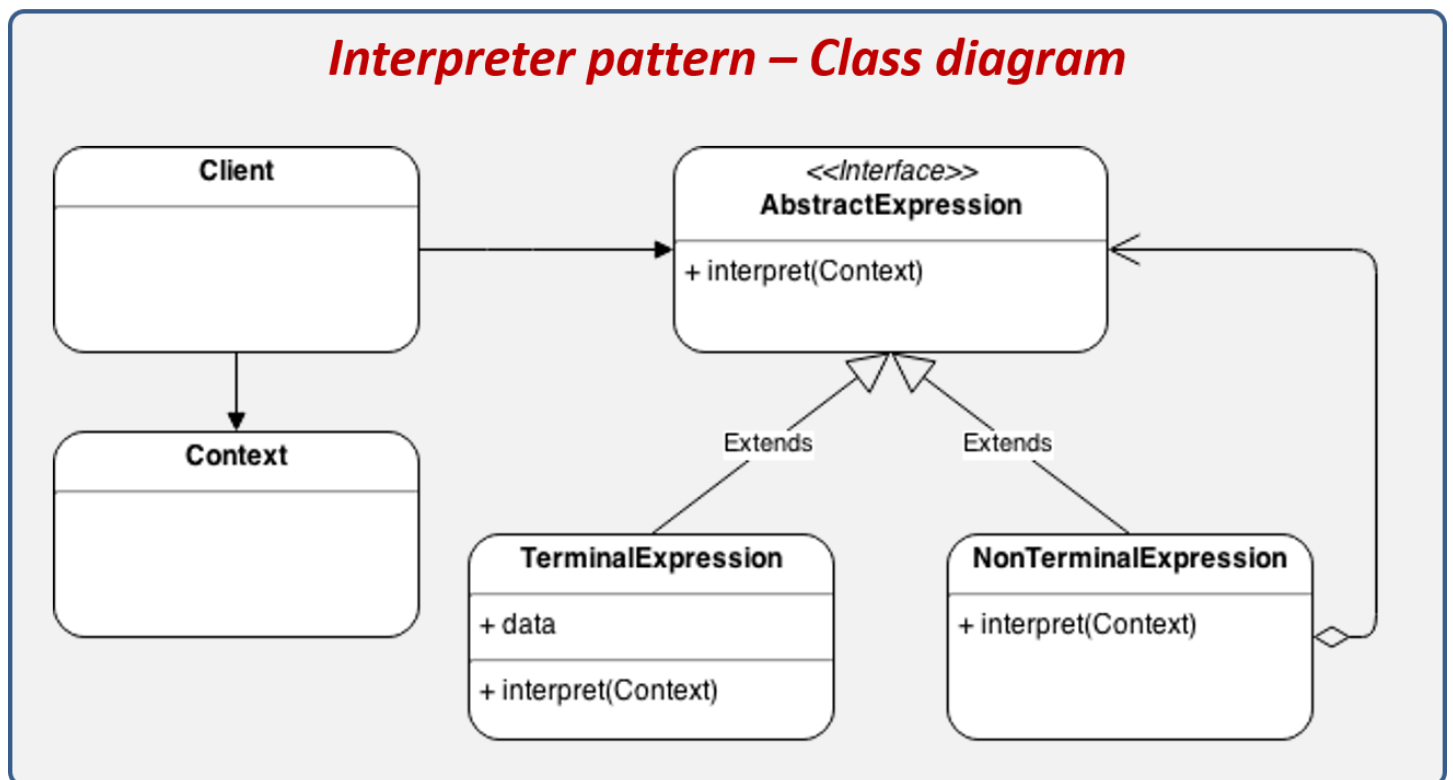
El proyecto esta desarrollado en el lenguaje de programacion python junto con ply que es una herramienta de análisis escrita exclusivamente en python. Es, en esencia, una reimplementación de lex y yacc originalmente en lenguaje c y se utiliza el patron interprete para poder realizar el siguiente proyecto.

FLUJO DEL PROGRAMA



PATRON UTILIZADO

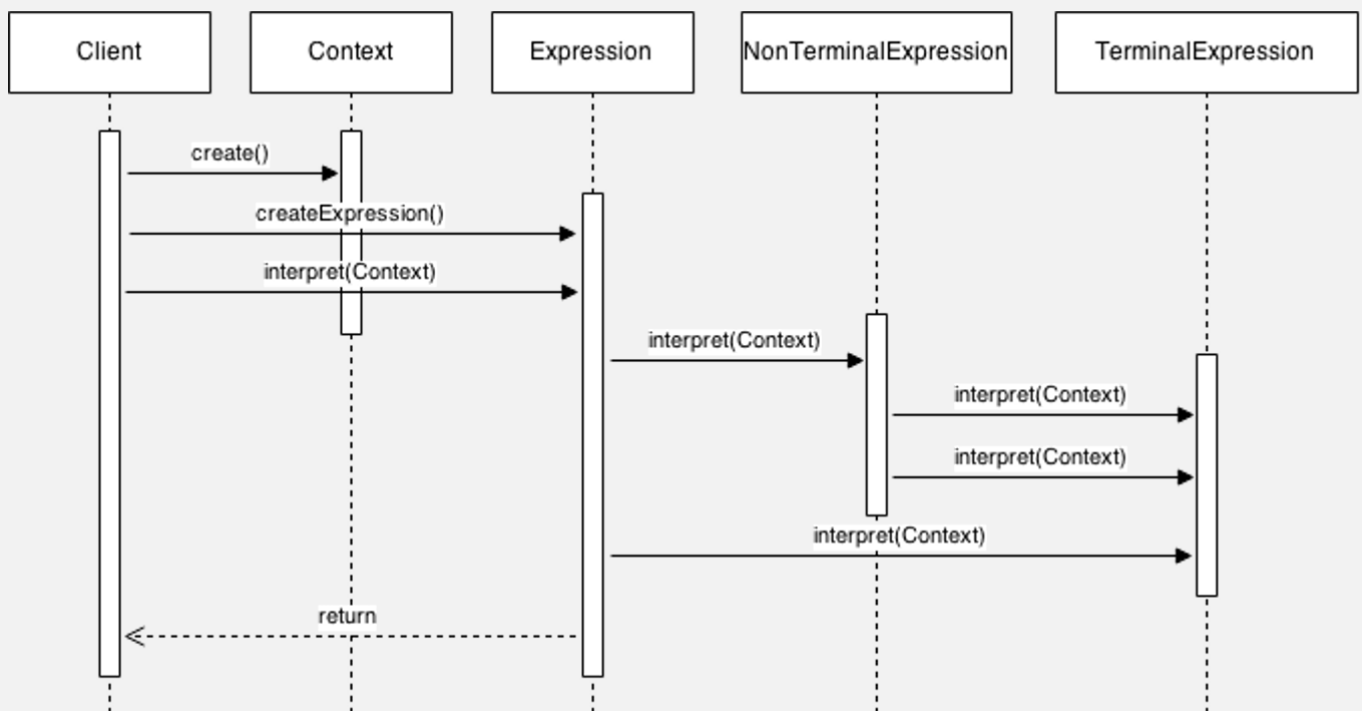
Patron interprete



El patrón de diseño interpreter es utilizado para evaluar un lenguaje definido como Expresiones, este patrón nos permite interpretar un lenguaje como Java, C#, SQL o incluso un lenguaje inventado por nosotros el cual tiene un significado; y darnos una respuesta tras evaluar dicho lenguaje.

Interpreter es uno de los patrones de diseño más complejos debido a que para su funcionalidad debe combinar técnicas de programación orientada a objetos avanzada y su interpretación puede ser algo confusa, las principales cosas con las que nos enfrentaremos son la Herencia, Polimorfismo y la Recursividad.


Interpreter pattern – Diagram of sequence

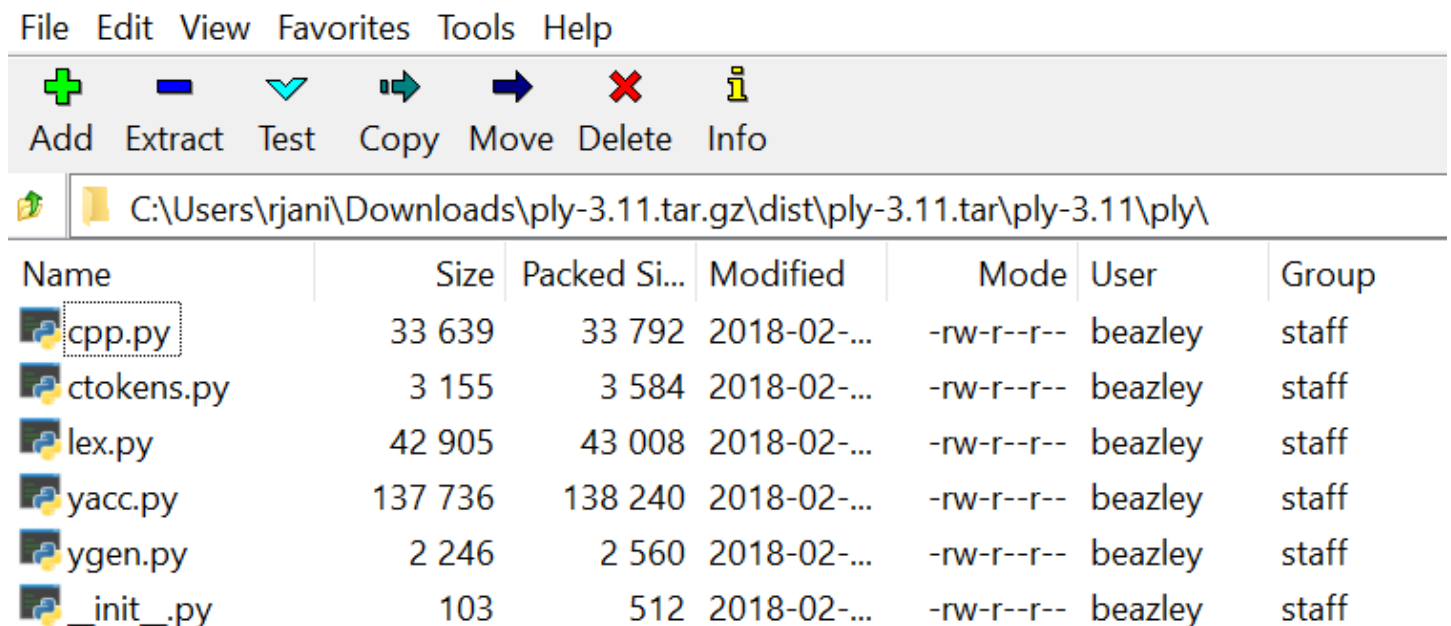


1. El *usuario* crea el *contexto* para la ejecución del interpreter.
2. El *usuario* crea u obtiene la expresión a evaluar.
3. El *usuario* solicita la interpretación de la expresión al *interpreter* y le envía el *contexto*.
4. La *Expresión* manda llamar a las *Expresiones No Terminales* que contiene.
5. La *Expresión No Terminal* manda llamar a todas las *Expresiones Terminales*.
6. La *Expresión Raíz* solicita la interpretación de una *Expresión Terminal*.
7. La *expresión* se evalúa por completo y se tiene un resultado de la interpretación de todas las *expresiones terminales y no terminales*.

INSTALACION DE PLY

Para hacer uso de PLY en nuestro proyecto no hacemos instalación como tal, lo que necesitamos es descargar el archivo `ply-3.11.tar.gz` (versión 3.11 al momento de escribir este tutorial) de la página oficial de [PLY](#) y lo que hacemos es copiar el folder "ply" a nuestro proyecto.

 C:\Users\rjani\Downloads\ply-3.11.tar.gz\dist\ply-3.11.tar\ply-3.11\ply\

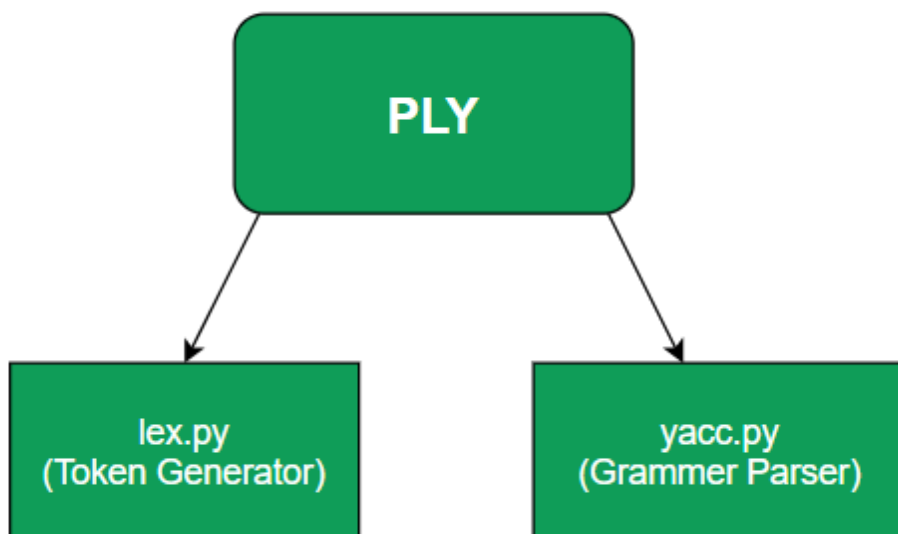


INTRODUCCION A PLY

PLY es una implementación en Python de `lex` y `yacc`, herramientas populares para la construcción de compiladores.

La principal tarea de un analizador léxico es leer los caracteres de entrada del programa fuente, agruparlos en *lexemas* y producir como salida una secuencia de *tokens*.

- Un *token* es un par que consiste en un nombre de token y un valor de atributo opcional.
- Un *lexema* es una secuencia de caracteres en el programa fuente, que coinciden con el patrón para un token y que el analizador léxico identifica como una instancia de este token.
- Un *patrón* es una descripción de la forma que pueden tomar los *lexemas* de un token.



El analizador sintáctico obtiene una cadena de tokens del analizador léxico y verifica que dicha cadena pueda generarse con la gramática para el lenguaje fuente. Una gramática proporciona una especificación precisa y fácil de entender de un lenguaje de programación.

En PLY se definen los patrones de los diferentes tokens que se desean reconocer, esto se hace a través de expresiones regulares. Mientras que las producciones y acciones para formar la gramática se definen a través de funciones.

GRAMATICA IMPLEMENTADA

```
reservadas = {  
  
    'boolean': 'BOOLEAN',  
    'true': 'TRUE',  
    'false': 'FALSE',  
    'order': 'ORDER',  
    'into': 'INTO',  
    'isnull': 'ISNULL',  
    'notnull': 'NOTNULL',  
    'replace': 'REPLACE',  
    'owner': 'OWNER',  
    'show': 'SHOW',  
    'databases': 'DATABASES',  
    'map' : 'MAP',  
    'list' : 'LIST',  
    'mode' : 'MODE',  
    'use' : 'USE',  
    'inherits': 'INHERITS',  
    'select' : 'SELECT',  
    'insert' : 'INSERT',  
    'update' : 'UPDATE',  
    'drop' : 'DROP',  
    'delete' : 'DELETE',  
    'alter' : 'ALTER',  
    'constraint' : 'CONSTRAINT',  
    'from' : 'FROM',
```

```
'group' : 'GROUP',
'by' : 'BY',
'where' : 'WHERE',
'having' : 'HAVING',
'create' : 'CREATE',
'type' : 'TYPE',
'primary' : 'PRIMARY',
'foreign' : 'FOREIGN',
'add' : 'ADD',
'rename' : 'RENAME',
'set' : 'SET',
'key' : 'KEY',
'if' : 'IF',
'else' : 'ELSE',
'unique' : 'UNIQUE',
'references' : 'REFERENCES',
'check' : 'CHECK',
'column' : 'COLUMN',
'database' : 'DATABASE',
'table' : 'TABLE',
'text' : 'TEXT',
'float' : 'FLOAT',
'values' : 'VALUES',
'int' : 'INT',
'default' : 'DEFAULT',
'null' : 'NULL',
'now' : 'NOW',
'smallint' : 'SMALLINT',
'integer' : 'INTEGER',
'bigint' : 'BIGINT',
'decimal' : 'DECIMAL',
'numeric' : 'NUMERIC',
'real' : 'REAL',
'double' : 'DOUBLE',
'precision' : 'PRECISION',
'money' : 'MONEY',
'varying' : 'VARYING',
'varchar' : 'VARCHAR',
'character' : 'CHARACTER',
'char' : 'CHAR',
'clear' : 'CLEAR',
'timestamp' : 'TIMESTAMP',
'date' : 'DATE',
'time' : 'TIME',
'interval' : 'INTERVAL',
'year' : 'YEAR',
'day' : 'DAY',
'hour' : 'HOUR',
'minute' : 'MINUTE',
'second' : 'SECOND',
'to' : 'TO',
'current_date' : 'CURRENT_DATE',
'current_time' : 'CURRENT_TIME',
'current_user' : 'CURRENT_USER',
'session_user' : 'SESSION_USER',
'date_part' : 'DATE_PART',
'month' : 'MONTH',
'enum' : 'ENUM',
'and' : 'AND',
'or' : 'OR',
'not' : 'NOT',
'between' : 'BETWEEN',
'unknown' : 'UNKNOWN',
'is' : 'IS',
'distinct' : 'DISTINCT',
```

```
'abs' : 'ABS',
'cbirt' : 'CBRT',
'ceil' : 'CEIL',
'ceiling' : 'CEILING',
'degrees' : 'DEGREES',
'extract' : 'EXTRACT',
'div' : 'DIV',
'exp' : 'EXP',
'trunc' : 'TRUNC',
'factorial' : 'FACTORIAL',
'floor' : 'FLOOR',
'gcd' : 'GCD',
'lcm' : 'LCM',
'ln' : 'LN',
'log' : 'LOG',
'log10' : 'LOG10',
'min_scale' : 'MIN_SCALE',
'mod' : 'MOD',
'pi' : 'PI',
'power' : 'POWER',
'radians' : 'RADIANS',
'round' : 'ROUND',
'scale' : 'SCALE',
'sign' : 'SIGN',
'sqrt' : 'SQRT',
'trim_scale' : 'TRIM_SCALE',
'truc' : 'TRUC',
'width_bucker' : 'WIDTH_BUCKET',
'random' : 'RANDOM',
'setseed' : 'SETSEED',
'contains' : 'CONTAINS',
'remove' : 'REMOVE',
'count' : 'COUNT',
'sum' : 'SUM',
'avg' : 'AVG',
'max' : 'MAX',
'min' : 'MIN',
'acos' : 'ACOS',
'acosc' : 'ACOSC',
'asin' : 'ASIN',
'asind' : 'ASIND',
'atan' : 'ATAN',
'atand' : 'ATAND',
'atan2' : 'ATAN2',
'atan2d' : 'ATAN2D',
'cos' : 'COS',
'cosd' : 'COSD',
'cot' : 'COT',
'cotd' : 'COTD',
'sin' : 'SIN',
'sind' : 'SIND',
'tan' : 'TAN',
'tand' : 'TAND',
'sinh' : 'SINH',
'cosh' : 'COSH',
'tanh' : 'TANH',
'asinh' : 'ASINH',
'acosh' : 'ACOSH',
'atanh' : 'ATANH',
'SIZE' : 'SIZE',

# FUNCIONES DE CADENA BINARIAS
'length' : 'LENGTH',
'substring' : 'SUBSTRING',
'trim' : 'TRIM',
```



```
'get_byte' : 'GET_BYTE',
'md5' : 'MD5',
'set_byte' : 'SET_BYTE',
'sha256' : 'SHA256',
'substr' : 'SUBSTR',
'convert' : 'CONVERT',
'encode' : 'ENCODE',
'decode' : 'DECODE',
'like' : 'LIKE',
'ilike' : 'ILIKE',
'similar' : 'SIMILAR',
'as' : 'AS',
'couter' : 'COUNTER',
'in' : 'IN',
'exists' : 'EXISTS',
'any' : 'ANY',
'all' : 'ALL',
'some' : 'SOME',
'join' : 'JOIN',
'inner' : 'INNER',
'left' : 'LEFT',
'right' : 'RIGHT',
'full' : 'FULL',
'outer' : 'OUTER',
'on' : 'ON',
'asc' : 'ASC',
'desc' : 'DESC',
'nulls' : 'NULLS',
'first' : 'FIRST',
'last' : 'LAST',
'case' : 'CASE',
'when' : 'WHEN',
'then' : 'THEN',
'end' : 'END',
'greatest' : 'GREATEST',
'least' : 'LEAST',
'limit' : 'LIMIT',
'offset' : 'OFFSET',
'union' : 'UNION',
'intersect' : 'INTERSECT',
'except' : 'EXCEPT',
'prueba' : 'PRUEBA'
}

def t_ENTERO(t):
    r'\d+'
    try:
        t.value = int(t.value)
    except ValueError:
        print("Integer value too large %d", t.value)
        t.value = 0
    return t

def t_TKDECIMAL(t):
    r'\d+\.\d+'
    try:
        t.value = float(t.value)
    except ValueError:
        print("Float value too large %d", t.value)
        t.value = 0
    return t

def t_CADENA(t):
    r'\'.*?\'
```

```

    t.value = t.value[1:-1]
    return t

def t_CADENADOBLE(t):
    r'\".*?\\"'
    t.value = t.value[1:-1]
    return t

def t_ID(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    t.type = reservadas.get(t.value.lower(), 'ID')
    return t

t_ignore = " \t"

def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value) # t.value.count("\n")

def t_COMENTARIO_SIMPLE(t):
    r'\--.*\n'
    t.lexer.lineno += 1

def t_COMENTARIO_MULTILINEA(t):
    r'/\*(.|\n)*?\*/'
    t.lexer.lineno += t.value.count('\n')

def t_error(t):
    print("Caracter NO Valido: '%s'" % t.value[0])
    t.lexer.skip(1)
    print(t.value[0])

precedence = (
    #('left', 'CONCAT'),
    #('left', 'MENOR', 'MAYOR', 'IGUAL', 'MENORIGUAL', 'MAYORIGUAL', 'DIFERENTE'),
    ('left', 'IGUAL', 'DISTINTO'),
    ('left', 'MENORQUE', 'MAYORQUE', 'MENORIG', 'MAYORIG'),
    ('left', 'MAS', 'MENOS'),
    ('left', 'MULTI', 'DIVISION', 'MODULO'),
    ('left', 'TKEXP'),
    #('right', 'UMENOS'),
)

def p_init(t):
    'init : sentences'
    t[0] = Arbol(t[1])

def p_sentences(t):
    '''
    sentences : sentences setInstruccions
              | setInstruccions
    '''
    if len(t) == 3:
        t[0] = t[1]
        t[0].append(t[2])
    else:
        t[0] = [t[1]]

def p_setInstruccions(t):
    '''
    setInstruccions : sentence PTCOMA
    '''
    t[0] = t[1]

def p_sentence(t):
    '''
    sentence : ddl

```

```
'''
t[0] = t[1]
def p_ddl_select(t):
'''
    ddl : select
'''
t[0] = t[1]

def p_ddl_use(t):
'''
    ddl : use_database
'''
t[0] = t[1]

def p_use_database(t):
'''
    use_database : USE ID
'''
t[0] = UseDatabase(t[2])

def p_ddl_table_create(t):
'''
    ddl : table_create
'''
t[0] = t[1]

def p_ddl_insert(t):
'''
    ddl : insert
'''
t[0]=t[1]

def p_ddl_update(t):
'''
    ddl : update
'''
t[0] = t[1]

def p_ddl_deletetable(t):
'''
    ddl : deletetable
'''
pass

def p_ddl_create_db(t):
'''
    ddl : create_db
'''
t[0] = t[1]

def p_ddl_alter_table(t):
'''
    ddl : alter_table
'''
pass

def p_ddl_create_type(t):
'''
    ddl : create_type
'''
pass

def p_ddl_alter_database(t):
```

```

'''
    ddl : alter_database
'''
t[0] = t[1]

def p_ddl_drop_database(t):
    '''
        ddl : drop_database
    '''
    t[0] = t[1]

def p_select(t):
    '''
        select : SELECT listavalores FROM listavalores listawhere
                | SELECT EXTRACT PARIZQ time FROM TIMESTAMP CADENA PARDER
                | SELECT DATE_PART PARIZQ CADENA COMA INTERVAL CADENA PARDER
                | SELECT NOW PARIZQ PARDER
                | SELECT CURRENT_DATE
                | SELECT CURRENT_TIME
                | SELECT TIMESTAMP CADENA
    '''
    if len(t) == 6:
        # SELECT listavalores FROM listavalores listawhere
        t[0] = select(t[2], t[4], t[5], 1, 1)
    elif t[2].lower() == "extract":
        # SELECT EXTRACT PARIZQ time FROM TIMESTAMP CADENA PARDER
        pass
    elif t[2].lower() == "date_part":
        # SELECT DATE_PART PARIZQ CADENA COMA INTERVAL CADENA PARDER
        pass
    elif t[2].lower() == "now":
        # SELECT NOW PARIZQ PARDER
        pass
    elif t[2].lower() == "current_date":
        # SELECT CURRENT_DATE
        pass
    elif t[2].lower() == "current_time":
        # SELECT CURRENT_TIME
        pass
    elif t[2].lower() == "timestamp":
        # SELECT TIMESTAMP CADENA
        pass

def p_select_simple(t):
    '''
        select : SELECT listavalores FROM listavalores
    '''
    # SELECT SIMPLE
    t[0] = select(t[2], t[4], "N/A", 1, 1)

def p_select_simple_simple(t):
    '''
        select : SELECT listavalores
    '''
    t[0] = select_simple_simple(t[2], 1, 1)

def p_time(t):
    '''
        time : YEAR
              | HOUR
              | SECOND
              | MINUTE
              | MONTH
              | DAY
    '''

```

```
t[0] = t[1]

def p_listawhere(t):
    '''
        listawhere : listawhere atributoselect
                   | atributoselect
    '''
    if len(t) == 3:
        t[0] = t[1]
        t[0].append(t[2])
    else:
        t[0] = [t[1]]

def p_atributoselectit(t):
    '''
        atributoselect : WHERE exp
                       | ORDER BY exp ordenamiento
                       | GROUP BY listavalores
                       | LIMIT exp
                       | HAVING exp
    '''
    if t[1].lower() == "where":
        t[0] = Condicion(t[2], "where", None, 1, 1)
    elif t[1].lower() == "order":
        # ORDER BY listavalores ordenamiento
        t[0] = Condicion(t[3], "ORDER", t[4], 1, 1)
        pass
    elif t[1].lower() == "group":
        # GROUP BY listavalores
        pass
    elif t[1].lower() == "limit":
        # LIMIT exp
        t[0] = Condicion(t[2], "LIMIT", None, 1, 1)
        pass
    elif t[1].lower() == "having":
        # HAVING exp
        t[0] = Condicion(t[2], "where", None, 1, 1)
        pass

def p_atributoselectit_subquery(t):
    '''
        atributoselect : subquery
    '''
    # subquery
    t[0] = t[1]

def p_ordenamiento(t):
    '''
        ordenamiento : ASC
                     | DESC
    '''
    t[0] = str(t[1])

def p_listavalores(t):
    '''
        listavalores : listavalores COMA exp
                     | exp
    '''
    if len(t) == 4:
        t[0] = t[1]
        t[0].append(t[3])
    else:
        t[0] = [t[1]]

def p_exp_count(t):
```

```
'''
    exp    : COUNT PARIZQ exp PARDER
            | COUNT PARIZQ MULTI PARDER
'''
if t[3]=='*':
    #COUNT PARIZQ MULTI PARDER
    pass
else:
    #COUNT PARIZQ exp PARDER
    pass
def p_exp_sum(t):
    '''
        exp    : SUM PARIZQ exp PARDER
    '''
    pass
def p_exp_avg(t):
    '''
        exp    : AVG PARIZQ exp PARDER
    '''
    pass
def p_exp_greatest(t):
    '''
        exp    : GREATEST PARIZQ listavalores PARDER
    '''
    pass
def p_exp_least(t):
    '''
        exp    : LEAST PARIZQ listavalores PARDER
    '''
    pass
def p_exp_max(t):
    '''
        exp    : MAX PARIZQ exp PARDER
    '''
    pass
def p_exp_min(t):
    '''
        exp    : MIN PARIZQ exp PARDER
    '''
    pass
def p_exp_abs(t):
    '''
        exp    : ABS PARIZQ exp PARDER
    '''
    pass
def p_exp_cbrt(t):
    '''
        exp    : CBRT PARIZQ exp PARDER
    '''
    pass
def p_exp_ceil(t):
    '''
        exp    : CEIL PARIZQ exp PARDER
    '''
    pass
def p_exp_ceiling(t):
    '''
        exp    : CEILING PARIZQ exp PARDER
    '''
    pass
def p_exp_degrees(t):
    '''
        exp    : DEGREES PARIZQ exp PARDER
    '''
```

```
pass

def p_exp_div(t):
    '''
        exp      : DIV PARIZQ exp COMA exp PARDER
    '''
    pass

def p_exp_tkexp(t):
    '''
        exp      : TKEXP PARIZQ exp PARDER
    '''
    pass

def p_exp_factorial(t):
    '''
        exp      : FACTORIAL PARIZQ exp PARDER
    '''
    t[0] = Select_simples(t[3], "FACTORIAL", 1, 1)
    pass

def p_exp_floor(t):
    '''
        exp      : FLOOR PARIZQ exp PARDER
    '''
    pass

def p_exp_gcd(t):
    '''
        exp      : GCD PARIZQ exp COMA exp PARDER
    '''
    pass

def p_exp_ln(t):
    '''
        exp      : LN PARIZQ exp PARDER
    '''
    pass

def p_exp_log(t):
    '''
        exp      : LOG PARIZQ exp PARDER
    '''
    pass

def p_exp_mod(t):
    '''
        exp      : MOD PARIZQ exp COMA exp PARDER
    '''
    pass

def p_exp_pi(t):
    '''
        exp      : PI PARIZQ PARDER
    '''
    pass

def p_exp_now(t):
    '''
        exp      : NOW PARIZQ PARDER
    '''
    pass

def p_exp_power(t):
    '''
        exp      : POWER PARIZQ exp COMA exp PARDER
    '''
    pass

def p_exp_radians(t):
    '''
        exp      : RADIANS PARIZQ exp PARDER
    '''
    pass

def p_exp_round(t):
```

```
'''
    exp    : ROUND PARIZQ exp PARDER
'''
pass
def p_exp_sign(t):
    '''
        exp    : SIGN PARIZQ exp PARDER
        '''
    pass
def p_exp_sqrt(t):
    '''
        exp    : SQRT PARIZQ exp PARDER
        '''
    pass
def p_exp_width(t):
    '''
        exp    : WIDTH_BUCKET PARIZQ exp COMA exp COMA exp COMA exp PARDER
        '''
    pass
def p_exp_trunc(t):
    '''
        exp    : TRUNC PARIZQ exp PARDER
        '''
    pass
def p_exp_random(t):
    '''
        exp    : RANDOM PARIZQ PARDER
        '''
    pass
def p_exp_acos(t):
    '''
        exp    : ACOS PARIZQ exp PARDER
        '''
    pass
def p_exp_acosd(t):
    '''
        exp    : ACOSD PARIZQ exp PARDER
        '''
    pass
def p_exp_asin(t):
    '''
        exp    : ASIN PARIZQ exp PARDER
        '''
    pass
def p_exp_asind(t):
    '''
        exp    : ASIND PARIZQ exp PARDER
        '''
    pass
def p_exp_atan(t):
    '''
        exp    : ATAN PARIZQ exp PARDER
        '''
    pass
def p_exp_atand(t):
    '''
        exp    : ATAND PARIZQ exp PARDER
        '''
    pass
def p_exp_atan2(t):
    '''
        exp    : ATAN2 PARIZQ exp COMA exp PARDER
        '''
    pass
```



```
def p_exp_atan2d(t):
    '''
        exp      : ATAN2D PARIZQ exp COMA exp PARDER
    '''
    pass
def p_exp_cos(t):
    '''
        exp      : COS PARIZQ exp PARDER
    '''
    pass
def p_exp_cosd(t):
    '''
        exp      : COSD PARIZQ exp PARDER
    '''
    pass
def p_exp_cot(t):
    '''
        exp      : COT PARIZQ exp PARDER
    '''
    pass
def p_exp_cotd(t):
    '''
        exp      : COTD PARIZQ exp PARDER
    '''
    pass
def p_exp_sin(t):
    '''
        exp      : SIN PARIZQ exp PARDER
    '''
    pass
def p_exp_sind(t):
    '''
        exp      : SIND PARIZQ exp PARDER
    '''
    pass
def p_exp_tan(t):
    '''
        exp      : TAN PARIZQ exp PARDER
    '''
    pass
def p_exp_tand(t):
    '''
        exp      : TAND PARIZQ exp PARDER
    '''
    pass
def p_exp_sinh(t):
    '''
        exp      : SINH PARIZQ exp PARDER
    '''
    pass
def p_exp_cosh(t):
    '''
        exp      : COSH PARIZQ exp PARDER
    '''
    pass
def p_exp_tanh(t):
    '''
        exp      : TANH PARIZQ exp PARDER
    '''
    pass
def p_exp_asinh(t):
    '''
        exp      : ASINH PARIZQ exp PARDER
    '''
    pass
```

```
def p_exp_acosh(t):
    '''
        exp      : ACOSH PARIZQ exp PARDER
    '''
    pass
def p_exp_atanh(t):
    '''
        exp      : ATANH PARIZQ exp PARDER
    '''
    pass
def p_exp_length(t):
    '''
        exp      : LENGTH PARIZQ exp PARDER
    '''
    pass
def p_exp_substring(t):
    '''
        exp      : SUBSTRING PARIZQ exp COMA exp COMA exp PARDER
    '''
    pass
def p_exp_trim(t):
    '''
        exp      : TRIM PARIZQ exp PARDER
    '''
    pass
def p_exp_md5(t):
    '''
        exp      : MD5 PARIZQ exp PARDER
    '''
    pass
def p_exp_sha256(t):
    '''
        exp      : SHA256 PARIZQ exp PARDER
    '''
    pass
def p_exp_substr(t):
    '''
        exp      : SUBSTR PARIZQ exp COMA exp COMA exp PARDER
    '''
    pass
def p_exp_getbyte(t):
    '''
        exp      : GET_BYTE PARIZQ exp COMA exp PARDER
    '''
    pass
def p_exp_setbyte(t):
    '''
        exp      : SET_BYTE PARIZQ exp COMA exp COMA exp PARDER
    '''
    pass
def p_exp_convert(t):
    '''
        exp      : CONVERT PARIZQ exp AS tipo PARDER
    '''
    pass
def p_exp_encode(t):
    '''
        exp      : ENCODE PARIZQ exp COMA exp PARDER
    '''
    pass
```

```
def p_exp_decode(t):
    '''
        exp : DECODE PARIZQ exp COMA exp PARDER
    '''
    pass
```

```
def p_exp_opunary(t):
    '''
        exp : ORBB exp
            | ORBBDOBLE exp
            | NOTBB exp
            | MAS exp
            | MENOS exp
            | NOT exp
            | IS exp
            | EXISTS exp
    '''
    if t[1]=='|':
        #ORBB exp
        pass
    elif t[1] == '||':
        # ORBBDOBLE exp
        pass
    elif t[1] == '~':
        # NOTBB exp
        pass
    elif t[1] == '+':
        # MAS exp
        pass
    elif t[1] == '-':
        # MENOS exp
        pass
    elif t[1] == 'not':
        # NOT exp
        pass
    elif t[1] == 'is':
        # IS exp
        pass
    elif t[1].lower() == 'exists':
        # IS exp
        pass
```

```
def p_exp_case(t):
    '''
        exp : case
    '''
    pass
```

```
def p_exp_between(t):
    '''
        exp : exp BETWEEN exp AND exp
    '''
    pass
```

```
def p_exp_distinct(t):
    '''
        exp : exp IS DISTINCT FROM exp
    '''
    pass
```

```
def p_exp_notdistinct(t):
    '''
        exp : exp IS NOT DISTINCT FROM exp
    '''
```

```

pass

def p_exp(t):
    '''
        exp      : exp ANDBB      exp
                  | exp ORBB      exp
                  | exp NUMERAL   exp
                  | exp SHIFTIZQ  exp
                  | exp SHIFTDER  exp
                  | exp TKEXP      exp
                  | exp MULTI      exp
                  | exp DIVISION   exp
                  | exp MODULO     exp
                  | exp MAS        exp
                  | exp MENOS      exp
                  | exp LIKE       exp
                  | exp ILIKE      exp
                  | exp SIMILAR    exp
                  | exp NOT        exp
                  | exp IN         exp
                  | exp IGUAL      exp
                  | exp MAYORQUE   exp
                  | exp MENORQUE   exp
                  | exp MAYORIG    exp
                  | exp MENORIG    exp
                  | exp IS         exp
                  | exp ISNULL     exp
                  | exp NOTNULL    exp
                  | exp AND        exp
                  | exp OR         exp
                  | expSimple
                  | exp NOT IN exp
    '''
    if len(t) == 4:
        #t[0] = Opera_Relacionales(t[1], t[3], "=", 1, 1)
        if t[2]=='&':
            #exp ANDBB exp
            t[0] = OperacionesLogicas(t[1], t[3], "&", 1, 1)
            pass
        elif t[2]=='|':
            # exp ORBB exp
            pass
        elif t[2]=='#':
            # exp NUMERAL exp
            pass
        elif t[2]=='<<':
            # exp SHIFTIZQ exp
            pass
        elif t[2]=='>>':
            # exp SHIFTDER exp
            pass
        elif t[2]=='^':
            # exp TKEXP exp
            pass
        elif t[2]=='*':
            # exp MULTI exp
            pass
        elif t[2]=='/':
            # exp DIVISION exp
            pass
        elif t[2]=='%':
            # exp MODULO exp
            pass
        elif t[2]=='+' :
            # exp MAS exp

```

```

        pass
    elif t[2]=='-':
        # exp MENOS exp
        pass
    elif t[2].lower()=='like':
        # exp like exp
        pass
    elif t[2].lower()=='similar':
        # exp ORBB exp
        pass
    elif t[2].lower()=='ilike':
        # exp ilike exp
        pass
    elif t[2].lower()=='not':
        # exp not exp
        pass
    elif t[2].lower()=='in':
        # exp IN exp
        pass
    elif t[2]=='=':
        # exp IGUAL exp
        t[0] = OperadoresCondicionales(t[1], t[3], "=")
        t[0] = Opera_Relacionales(t[1], t[3], "=", 1, 1)
        pass
    elif t[2]=='>':
        # exp MAYORQUE exp
        t[0] = OperadoresCondicionales(t[1], t[3], ">")
    elif t[2]=='<':
        # exp MENORQUE exp
        t[0] = OperadoresCondicionales(t[1], t[3], "<")
    elif t[2]=='>=':
        # exp MAYORIG exp
        t[0] = OperadoresCondicionales(t[1], t[3], ">=")
    elif t[2]=='<=':
        # exp MENO
        t[0] = OperadoresCondicionales(t[1], t[3], "<=")
    elif t[2].lower()=='is':
        # exp IS exp
        pass
    elif t[2].lower()=='isnull':
        # exp ISNULL exp
        pass
    elif t[2].lower()=='notnull':
        # exp NOTNULL exp
        pass
    elif t[2].lower()=='and':
        # exp AND exp
        pass
    elif t[2].lower()=='or':
        # exp OR exp
        pass
    pass
elif len(t) == 5:
    #exp NOT IN exp
    pass
elif len(t) == 2:
    #expSimple
    t[0] = t[1]
def p_expSimples(t):
    '''
        expSimple      : NULL
                        | subquery
                        | DISTINCT exp
    '''
    t[0] = t[1]

```

```
def p_expSimples_ACCESO_TYPE(t):
    '''
        expSimple : ID PARIZQ exp PARDER
    '''
    t[0] = indexador_auxiliar(t[1], t[3], 7)

def p_expSimples_ALIAS_MULTI(t):
    '''
        expSimple : ID PT MULTI
    '''
    t[0] = indexador_auxiliar(t[1], "MULTI", 6)

def p_expSimples_MULTI(t):
    '''
        expSimple : MULTI
    '''
    t[0] = indexador_auxiliar("GLOBAL", "MULTI", 5)

def p_expSimples_ID(t):
    '''
        expSimple : ID
    '''
    t[0] = indexador_auxiliar(t[1], t[1], 4)

def p_expSimples_ID_PT_ID(t):
    '''
        expSimple : ID PT ID
    '''
    t[0] = indexador_auxiliar(t[1], t[3], 3)

def p_expSimples_ID_ID(t):
    '''
        expSimple : ID ID
    '''
    t[0] = indexador_auxiliar(t[1], t[2], 1)

def p_expSimples_exp_AS_ID(t):
    '''
        expSimple : ID AS ID
                    | exp AS CADENA
                    | exp AS ID
    '''
    t[0] = indexador_auxiliar(t[1], t[3], 1)

def p_subquery(t):
    '''
        subquery : PARIZQ select PARDER
                  | PARIZQ select PARDER ID
                  | PARIZQ select PARDER AS ID
    '''
    if len(t) == 4:
        #PARIZQ select PARDER
        pass
    elif len(t) == 5:
        #PARIZQ select PARDER ID
        t[0] = indexador_auxiliar(t[2], t[4], 2)
    elif len(t) == 6:
        #PARIZQ select PARDER AS ID
        t[0] = indexador_auxiliar(t[2], t[5], 2)
        pass

def p_case(t):
    '''
```

```

        case : CASE WHEN exp THEN exp lista_when ELSE exp END
              | CASE WHEN exp THEN exp lista_when END
              | CASE WHEN exp THEN exp ELSE exp END
              | CASE WHEN exp THEN exp END
    '''
    # t[0] = interprete

def p_lista_when(t):
    '''
        lista_when : lista_when when_else
                   | when_else
    '''
    if len(t) == 3:
        t[0] = t[1]
        t[0].append(t[2])
    else:
        t[0] = [t[1]]

def p_when_else(t):
    '''
        when_else : WHEN exp THEN exp
    '''

def p_expSimples_entero(t):
    '''
        expSimple : ENTERO
    '''
    t[0] = ENTERO(t[1],1,1)

def p_expSimples_decimal(t):
    '''
        expSimple : TKDECIMAL
    '''
    t[0] = DECIMAL(t[1],1,1)

def p_expSimples_cadenas(t):
    '''
        expSimple : CADENA
    '''
    t[0] = CADENAS(t[1],1,1)

def p_expSimples_cadenadoble(t):
    '''
        expSimple : CADENADOBLE
    '''
    t[0] = CADENAS(t[1],1,1)

def p_expSimples_true(t):
    '''
        expSimple : TRUE
    '''
    t[0] = BOOLEANO(True,1,1)

def p_expSimples_false(t):
    '''
        expSimple : FALSE
    '''
    t[0] = BOOLEANO(False,1,1)

def p_table_create(t):
    '''
        table_create : CREATE TABLE ID PARIZQ lista_table COMA listadolprimary inherits
                    | CREATE TABLE ID PARIZQ lista_table inherits
                    | CREATE TABLE IF NOT EXISTS ID PARIZQ lista_table COMA
listadolprimary inherits
                    | CREATE TABLE IF NOT EXISTS ID PARIZQ lista table inherits

```

```

'''
if len(t)==9:
    # CREATE TABLE ID PARIZQ lista_table COMA listadolprimary inherits
    t[0] = CreateTable(t.lineno, 0, t[3], t[5], t[7], t[8])
if len(t)==7:
    # CREATE TABLE ID PARIZQ lista_table inherits
    t[0] = CreateTable(t.lineno, 0, t[3], t[5], None, t[6])
if len(t)==12:
    t[0] = CreateTable(t.lineno, 0, t[6], t[8], t[10], t[11])
if len(t)==10:
    #CREATE TABLE IF NOT EXISTS ID PARIZQ lista_table inherits
    t[0] = CreateTable(t.lineno, 0, t[6], t[8], None, t[9])

#todo:no se que hacer con PARDER FALTA ? O SOLO ASI ES ?
def p_inherits(t):
    '''
        inherits : PARDER INHERITS PARIZQ ID PARDER
    '''
    t[0] = clases_auxiliares.Inherits(t[4])

def p_inherits_parder(t):
    '''
        inherits : PARDER
    '''
    t[0] = None

def p_lista_table(t):
    '''
        lista_table : lista_table COMA atributo_table
                    | atributo_table
    '''
    if len(t) == 4:
        t[0] = t[1]
        t[0].append(t[3])
    else:
        t[0] = [t[1]]

def p_listadolprimary(t):
    '''
        listadolprimary : listadolprimary COMA lista_primary
                        | lista_primary
    '''
    if len(t) == 4:
        t[0] = t[1]
        t[0].append(t[3])
    else:
        t[0] = [t[1]]

def p_lista_primary(t):
    '''
        lista_primary : PRIMARY KEY PARIZQ listaids PARDER
                    | FOREIGN KEY PARIZQ listaids PARDER REFERENCES ID PARIZQ listaids
                    | CONSTRAINT ID CHECK PARIZQ exp PARDER
                    | CHECK PARIZQ exp PARDER
                    | UNIQUE PARIZQ listaids PARDER
    '''
    if len(t)==6:
        t[0] = clases_auxiliares.PrimaryKeyC(t[4])
    elif len(t)==11:
        references = clases_auxiliares.References(t[7], t[9])
        t[0] = clases_auxiliares.ForeignKeyC(references, t[4])

```



```

elif len(t)==7:
    constraint = clases_auxiliares.Constraint(t[2])
    t[0] = clases_auxiliares.Check(t[5], constraint)
elif t[1].lower() == 'check':
    t[0] = clases_auxiliares.Check(t[3])
elif t[1].lower() == 'unique':
    t[0] = clases_auxiliares.UniqueC(t[3])

def p_atributo_table(t):
    '''
        atributo_table : ID tipocql listaespecificaciones
                        | ID tipocql
    '''
    if len(t)==4:
        t[0] = clases_auxiliares.Columna(t[1], t[2], t[3])
    elif len(t)==3:
        t[0] = clases_auxiliares.Columna(t[1], t[2])
def p_listaespecificaciones(t):
    '''
        listaespecificaciones : listaespecificaciones especificaciones
                        | especificaciones
    '''
    if len(t) == 3:
        t[0] = t[1]
        t[0].append(t[2])
    else:
        t[0] = [t[1]]

def p_especificaciones(t):
    '''
        especificaciones : DEFAULT exp
                        | PRIMARY KEY
                        | REFERENCES ID
                        | CONSTRAINT ID UNIQUE
                        | CONSTRAINT ID CHECK PARIZQ exp PARDER
                        | CHECK PARIZQ exp PARDER
                        | UNIQUE
                        | NOT NULL
                        | NULL
    '''

    if t[1].lower() == 'default':
        t[0] = clases_auxiliares.Default(t[2])
    elif t[1].lower() == 'primary':
        t[0] = clases_auxiliares.PrimaryKey()
    elif t[1].lower() == 'references':
        t[0] = clases_auxiliares.References(t[2])
    elif len(t) == 4:
        constraint = clases_auxiliares.Constraint(t[2])
        t[0] = clases_auxiliares.Unique(constraint)
    elif len(t) == 7:
        constraint = clases_auxiliares.Constraint(t[2])
        t[0] = clases_auxiliares.Check(t[5], constraint)
    elif t[1].lower() == 'check':
        t[0] = clases_auxiliares.Check(t[3])
    elif t[1].lower() == 'unique':
        t[0] = clases_auxiliares.Unique()
    elif t[1].lower() == 'not':
        t[0] = clases_auxiliares.NotNull()
    elif t[1].lower() == 'null':
        t[0] = clases_auxiliares.Null()
def p_tipocql(t):
    '''

```

```

        tipocql : tipo
'''
t[0] = t[1]
def p_tipocql_id(t):
'''
        tipocql : ID
'''
t[0] = t[1]
def p_tipo(t):
'''
        tipo : SMALLINT
            | INTEGER
            | BIGINT
            | DECIMAL
            | NUMERIC
            | REAL
            | MONEY
            | TEXT
            | TIME
            | DATE
            | TIMESTAMP
            | INTERVAL
            | BOOLEAN
            | DOUBLE PRECISION
            | CHARACTER VARYING PARIZQ exp PARDER
            | VARCHAR PARIZQ exp PARDER
            | CHAR PARIZQ exp PARDER
'''
if t[1].lower() == 'integer':
    t[0] = '0'
elif t[1].lower() == 'smallint':
    t[0] = '4'
elif t[1].lower() == 'bigint':
    t[0] = '5'
elif t[1].lower() == 'decimal':
    t[0] = '1'
elif t[1].lower() == 'numeric':
    t[0] = '6'
elif t[1].lower() == 'real':
    t[0] = '7'
elif t[1].lower() == 'money':
    t[0] = '8'
elif t[1].lower() == 'text':
    t[0] = '9'
elif t[1].lower() == 'time':
    t[0] = '11'
elif t[1].lower() == 'date':
    t[0] = '12'
elif t[1].lower() == 'timestamp':
    t[0] = '13'
elif t[1].lower() == 'interval':
    t[0] = '14'
elif t[1].lower() == 'boolean':
    t[0] = '3'
elif t[1].lower() == 'double':
    t[0] = '15'
elif len(t) == 6:
    t[0] = '16'
elif t[1].lower() == 'varchar':
    t[0] = '18'
elif t[1].lower() == 'char':
    t[0] = '19'

def p_tipo_character_varying(t):

```

```

'''
    tipo : CHARACTER PARIZQ exp PARDER
'''
t[0] = '17'
def p_insert(t):
'''
    insert : INSERT INTO ID                               VALUES PARIZQ listavalores PARDER
           | INSERT INTO ID PARIZQ listaids PARDER VALUES PARIZQ listavalores PARDER
'''
if len(t)==8:
    #INSERT INTO ID VALUES PARIZQ listavalores PARDER
    t[0]= Insert(t[3],[],t[6],t.lineno,0)
elif len(t)==11:
    #INSERT INTO ID PARIZQ listaids PARDER VALUES PARIZQ listavalores PARDER
    t[0]= Insert(t[3],t[5],t[9],t.lineno,0)

def p_listaids(t):
'''
    listaids : listaids COMA ID
             | ID
'''
if len(t) == 4:#listaids COMA ID
    t[0] = t[1]
    t[0].append(t[3])
else:#ID
    t[0] = [t[1]]
def p_update(t):
'''
    update : UPDATE ID SET listaupdate WHERE exp
           | UPDATE ID SET listaupdate
'''
if len(t)==7:
    #UPDATE ID SET listaupdate WHERE exp
    t[0] = Update(t.lineno, 0, t[2], t[4], t[6])
elif len(t)==5:
    #UPDATE ID SET listaupdate
    pass

def p_listaupdate(t):
'''
    listaupdate : listaupdate COMA asignacionupdate
                | asignacionupdate
'''
if len(t) == 4:#listaupdate COMA asignacionupdate
    t[0] = t[1]
    t[0].append(t[3])
else:#asignacionupdate
    t[0] = [t[1]]
def p_asignacionupdate(t):
'''
    asignacionupdate : ID IGUAL exp
'''
t[0] = Opera_Relacionales(t[1], t[3], "u=", 1, 1)
def p_acceso(t):
'''
    acceso : acceso PT funcioncollection
           | acceso CORIZQ exp CORDER
           | ID
'''
if len(t)==4:
    #acceso PT funcioncollection
    pass
elif len(t)==5:
    #acceso CORIZQ exp CORDER
    pass

```

```

elif len(t)==2:
    #ID
    pass
def p_acceso_ID(t):
    '''
        acceso : acceso PT ID
    '''
def p_funcioncollection(t):
    '''
        funcioncollection : INSERT PARIZQ exp COMA exp PARDER
                          | INSERT PARIZQ exp PARDER
                          | SET PARIZQ exp COMA exp PARDER
                          | REMOVE PARIZQ exp PARDER
                          | SIZE PARIZQ PARDER
                          | CLEAR PARIZQ PARDER
                          | CONTAINS PARIZQ exp PARDER
                          | LENGTH PARIZQ PARDER
                          | SUBSTRING PARIZQ exp COMA exp PARDER
    '''
    if t[1].lower() == 'insert':
        if len(t)==7:
            #INSERT PARIZQ exp COMA exp PARDER
            pass
        elif len(t)==5:
            #INSERT PARIZQ exp PARDER
            pass
    elif t[1].lower() == 'set':
        #SET PARIZQ exp COMA exp PARDER
        pass
    elif t[1].lower() == 'remove':
        #REMOVE PARIZQ exp PARDER
        pass
    elif t[1].lower() == 'size':
        #SIZE PARIZQ PARDER
        pass
    elif t[1].lower() == 'clear':
        #CLEAR PARIZQ PARDER
        pass
    elif t[1].lower() == 'contains':
        #CONTAINS PARIZQ exp PARDER
        pass
    elif t[1].lower() == 'length':
        #LENGTH PARIZQ PARDER
        pass
    elif t[1].lower() == 'substring':
        #SUBSTRING PARIZQ exp COMA exp PARDER
        pass
def p_deletetable(t):
    '''
        deletetable : DELETE FROM ID WHERE exp
                   | DELETE FROM ID
                   | DELETE listaatributos FROM ID WHERE exp
                   | DELETE listaatributos FROM ID
    '''
    if len(t)==6:
        #DELETE FROM ID WHERE exp
        pass
    elif len(t) == 4:
        #DELETE FROM ID
        pass
    elif len(t) == 7:
        #DELETE listaatributos FROM ID WHERE exp
        pass
    elif len(t) == 5:
        # DELETE listaatributos FROM ID

```

```

        pass
def p_listaatributos(t):
    '''
        listaatributos : listaatributos COMA acceso
                        | acceso
    '''
    if len(t) == 4: #listaatributos COMA acceso
        t[0] = t[1]
        t[0].append(t[3])
    else: #acceso
        t[0] = [t[1]]
def p_create_db(t):
    '''
        create_db : CREATE OR REPLACE DATABASE IF NOT EXISTS createdb_extra
                  | CREATE OR REPLACE DATABASE createdb_extra
                  | CREATE DATABASE IF NOT EXISTS createdb_extra
                  | CREATE DATABASE createdb_extra
    '''
    if len(t) == 9:
        #CREATE OR REPLACE DATABASE IF NOT EXISTS createdb_extra
        t[0] = CreateDatabase(t[8], True, True)
    elif len(t) == 6:
        #CREATE OR REPLACE DATABASE createdb_extra
        t[0] = CreateDatabase(t[5], True, False)
    elif len(t) == 7:
        #CREATE DATABASE IF NOT EXISTS createdb_extra
        t[0] = CreateDatabase(t[6], False, True)
    elif len(t) == 4:
        #CREATE DATABASE createdb_extra
        t[0] = CreateDatabase(t[3], False, False)
def p_createdb_extra(t):
    '''
        createdb_extra : ID OWNER IGUAL exp MODE IGUAL exp
                       | ID OWNER IGUAL exp MODE exp
                       | ID OWNER exp MODE IGUAL exp
                       | ID OWNER exp MODE exp
                       | ID OWNER IGUAL exp
                       | ID MODE IGUAL exp
                       | ID OWNER exp
                       | ID MODE exp
                       | ID
    '''
    t[0] = t[1]
def p_drop_table(t):
    '''
        drop_table : DROP TABLE IF EXISTS ID
                   | DROP TABLE ID
    '''
    if len(t) == 6:
        #DROP TABLE IF EXISTS ID
        pass
    elif len(t) == 3:
        #DROP TABLE ID
        pass
def p_alter_table(t):
    '''
        alter_table : ALTER TABLE ID ADD listaespecificaciones
                   | ALTER TABLE ID DROP listaespecificaciones
                   | ALTER TABLE ID listacolumn
    '''
    if len(t) == 6:
        if t[4].lower() == 'add':
            #ALTER TABLE ID ADD listaespecificaciones
            pass

```

```

        elif t[4].lower() == 'drop':
            #ALTER TABLE ID DROP listaespecificaciones
            pass
    elif len(t)==5:
        #ALTER TABLE ID listacolumn
        pass
def p_listacolumn(t):
    '''
        listacolumn : listacolumn COMA column
                    | column
    '''
    if len(t) == 4:
        #listacolumn COMA column
        t[0] = t[1]
        t[0].append(t[3])
    else:
        #column
        t[0] = [t[1]]
def p_column(t):
    '''
        column : ALTER COLUMN ID listaespecificaciones
                | ADD COLUMN ID tipo
                | DROP COLUMN ID
    '''
    if t[1].lower()=='alter':
        #ALTER COLUMN ID listaespecificaciones
        pass
    elif t[1].lower() == 'add':
        #ADD COLUMN ID tipo
        pass
    elif t[1].lower() == 'drop':
        #DROP COLUMN ID
        pass
def p_create_type(t):
    '''
        create_type : CREATE TYPE ID AS ID PARIZQ listavalores PARDER
    '''
    # t[0] = interprete
    t[0] = type(t[3], t[7], 1, 1)
def p_alter_database(t):
    '''
        alter_database : ALTER DATABASE ID RENAME TO ID
                        | ALTER DATABASE ID OWNER TO CURRENT_USER
                        | ALTER DATABASE ID OWNER TO SESSION_USER
    '''
    if t[4].lower()=='rename':
        t[0] = AlterDatabase(t[3], t[6])
    elif t[4].lower()=='owner':
        #ALTER DATABASE ID OWNER TO ID <- aqui no hay progra xd
        pass
def p_drop_database(t):
    '''
        drop_database : DROP DATABASE IF EXISTS ID
                       | DROP DATABASE ID
    '''
    if len(t)==6:
        #DROP DATABASE IF EXISTS ID
        pass
    elif len(t) == 4:
        #DROP DATABASE ID
        pass

```

