Grammaire du langage CQL (Cassandra Query Language)

Les mots clés :

ADD	DISTINCT	LIMIT	STATIC
AGGREGATE	DOUBLE	LIST	STORAGE
ALL	DROP	LOGIN	STYPE
ALLOW	ENTRIES	MAP	SUPERUSER
ALTER	EXECUTE	MODIFY	TABLE
AND	EXISTS	NAN	TEXT
APPLY	FILTERING	NOLOGIN	TIME
AS	FINALFUNC	NORECURSIVE	TIMESTAMP
ASC	FLOAT	NOSUPERUSER	TIMEUUID
ASCII	FROM	NOT	TINYINT
AUTHORIZE	FROZEN	NULL	TO
BATCH	FULL	OF	TOKEN
BEGIN	FUNCTION	ON	TRIGGER
BIGINT	FUNCTIONS	OPTIONS	TRUNCATE
BLOB	GRANT	OR	TTL
BOOLEAN	IF	ORDER	TUPLE
BY	IN	PASSWORD	TYPE
CALLED	INDEX	PERMISSION	UNLOGGED
CLUSTERING	INET	PERMISSIONS	UPDATE
COLUMNFAMILY	INFINITY	PRIMARY	USE
COMPACT	INITCOND	RENAME	USER
CONTAINS	INPUT	REPLACE	USERS
COUNT	INSERT	RETURNS	USING
COUNTER	INT	REVOKE	UUID
CREATE	INTO	ROLE	VALUES
CUSTOM	JSON	ROLES	VARCHAR
DATE	KEY	SCHEMA	VARINT
DECIMAL	KEYS	SELECT	WHERE
DELETE	KEYSPACE	SET	WITH
DESC	KEYSPACES	SFUNC	WRITETIME
DESCRIBE	LANGUAGE	SMALLINT	

```
cassandra_statement ::= statement ;
statement ::= ddl_statement | dml_statement | dcl_statement
ddl_statement ::= create_statement
                | use_statement
                | alter_statement
                | drop_statement
                | truncate_statement
dml_statement ::= select_statement
                | insert_statement
                | update_statement
                | delete_statement
                | batch_statement
dcl_statement ::= grant_statement
                | revoke_statement
                | list_statement

create_statement ::= CREATE create_statement_ox ;
create_statement_ox ::= KEYSPACE create_keyspace_statement
                      | create_table_ox create_table_statement
                      | custom_ox INDEX create_index_statement
                      | MATERIALIZED create_materialized_statement
                      | ROLE create_role_statement
                      | USER create_user_statement
                      | or_replace_ox create_function_aggregate
                      | TYPE create_type_statement
                      | TRIGGER create_trigger_statement

create_table_ox ::= TABLE | COLUMNFAMILY
or_replace_ox ::= OR REPLACE | ε
create_function_aggregate ::= FUNCTION create_function_statement
                            | AGGREGATE create_aggregate_statement
custom_ox ::= CUSTOM | ε
```

keyspace:

```
create_keyspace_statement ::= is_not_exists keyspace_name WITH options
if_not_exists ::= IF NOT EXISTS | ε
options ::= option ( AND option )*


option ::= identifier = option_aux
option_aux ::= identifier | constant | map_literal
```

```
constant ::= string | integer | float | boolean | uuid | blob | NULL
map_literal ::= { map_literal_aux | ε }

term ::= { aux_1 }
        | [ aux_2 ]
        | ( aux_3
        | identifier ( aux_4 )
        | constant
        | ?
        | : identifier
aux_1 ::= opt_data aux_1_1
        | identifier aux_1_2
        | ε
opt_data ::= { aux_1 }
             | [ aux_2 ]
             | ( aux_3
             | constant
             | ?
             | : identifier
aux_1_1 ::= : term ( , term : term )*
          | ( , term )*
aux_1_2 ::= : term ( , identifier : term )*
          | ( aux_4 )
aux_2 ::= term ( , term )* | ε
aux_3 ::= term ( , term )* )    |    cql_type ) term
aux_4 ::= term ( , term )* | ε
```

Table:

```
create_table_statement ::= if_not_exists table_name
                           ( column_definition (, column_definition)*
                           primary_key_ox_para )
                           with_property_ox
table_name ::= keyspace_name_ox name
keyspace_name_ox ::= keyspace. | ε
name ::= unquoted_name | quoted_name
column_definition ::= identifier cql_type static_ox primary_key_ox
static_ox ::= STATIC | ε
primary_key_ox ::= PRIMARY KEY | ε
primary_key_ox_para ::= , PRIMARY KEY ( partition_key clustering_columns )
                           | ε
partition_key ::= identifier | ( identifier (, identifier)* )
clustering_columns ::= clustering_columns_ox | ε
```

```
clustering_columns_ox ::= , identifier (, identifier)*
with_property_ox ::= WITH table_option | ε
table_options ::= COMPACT STORAGE table_option_ox
                | CLUSTERING ORDER BY   ( clustering_order ) table_option_ox
                | options
table_option_ox ::= table_option | ε
clustering_order ::= identifier asc_desc ( , identifier asc_desc )*
asc_desc ::= ASC | DESC
```

Index:

```
create_index_statement ::= if_not_exists index_name_ox
                                ON table_name ( index_identifier ) using_ox
index_name_ox ::= idf_index {[a-zA-Z_0..9]*} | ε
index_identifier ::= idf_column | option_ox ( idf_column )
option_ox ::= KEYS | VALUES | ENTRIES | FULL
using_ox ::= using_statement | ε
using_statement ::= USING string_token index_options_ox
index_option_ox ::= index_options | ε
index_option ::= WITH OPTIONS = map_literal
```

materialized:

```
create_materialized_statement ::= VIEW if_not_exists idf_view
                                  AS select_statement
                                  PRIMARY KEY ( primary_key )
                                  WITH table_options
idf_view ::= {a-zA-Z_0..9}+
```

```
create_role_statement ::= if_not_exists idf_role role_option_ox
role_option_ox ::= WITH role_options | ε
role_options ::= role_op ( AND role_op )*
role_op ::= PASSWORD = string
          | LOGIN = boolean
          | SUPERUSER = boolean
          | OPTIONS = map_literal
```

```
create_user_statement ::= if_not_exists idf_role with_pass_ox user_op_ox
with_pass_ox ::= WITH PASSWORD string | ε
user_op_ox ::= SUPERUSER | NOSUPERUSER | ε
```

```
create_function_statement ::= if_not_exists idf_function ( arg_dec )
                                  called_ox ON NULL INPUT
                                  RETURNS cql_type
                                  LANGUAGE identifier
                                  AS string
agr_dec ::= identifier cql_type ( , identifier cql_type )*
called_ox ::= CALLED   | RETURNS NULL | ε


create_type_statement ::= if_not_exists udt_name
                                ( field_definition ( , field_definition)* )
field_definition ::= identifier cql_type

create_trigger_statement ::= if_not_exists trigger_name
                                ON table_name
                                USING string

use :
use_statement ::= USE keyspace_name

drop:

drop_statement ::= DROP drop_ox
drop_ox ::= KEYSPACE drop_keyspace_statement
            | TABLE drop_table_statement
            | INDEX drop_index_statement
            | ROLE drop_role_statement
            | USER drop_user_statement
            | FUNCTION drop_function_statement
            | AGGREGATE drop_aggregate_statement
            | TYPE drop_type_statement
            | TRIGGER drop_trigger_statement
            | MATERIALIZED drop_materialized_statement

drop_keyspace_statement ::= if_exists keyspace_name
drop_table_statement ::= if_exists table_name
drop_index_statement ::= if_exists index_name
drop_role_statement ::= if_exists role_name
drop_user_statement ::= if_exists role_name
drop_function_statement ::= if_exists function_name arg_sin
arg_sin ::= arg_sin_ox | ε
arg_sin_ox ::= cql_type ( , cql_type)*
drop_aggregate_statement ::= if_exists function_name arg_sin
```

```
drop_type_statement ::= if_exists udt_name
drop_trigger_statement ::= if_exists trigger_name ON table_name
drop_materialized_statement ::= VIEW if_exists view_name
if_exists ::= IF EXISTS | ε
```

truncate:

```
truncate_statement ::= TRUNCATE tab_ox table_name
tab_ox ::= TABLE | COLUMNSFAMILY | ε
```

alter :

```
alter_statement ::= ALTER alter_aux
alter_aux ::= KEYSPACE alter_keyspace_statement
            | TABLE alter_table_statement
            | ROLE alter_role_statement
            | USER alter_user_statement
            | TYPE alter_user_statement
alter_keyspace_statement ::= keyspace_name WITH options
alter_table_statement ::= table_name alter_table_instruction
alter_table_instruction ::= ALTER column_name TYPE cql_type
                        | ADD column_name cql_type ( , column_name cql_type )*
                        | DROP column_name ( column_name )*
                        | WITH options
alter_role_statement ::= role_name WITH role_options
alter_user_statement ::= role_name alter_user_aux
alter_user_aux ::= WITH PASSWORD string_token | user_option
user_option ::= SUPERUSER | NOSUPERUSER
alter_type_statement ::= udt_name alter_type_modification
alter_type_modification ::= ALTER identifier TYPE cql_type
                        | ADD field_definition
                        | RENAME identifier TO identifier (identifier TO identifier )*
field_definition ::= identifier cql_type
```

select:

```
select_statement ::= SELECT   option_json_distinct   select_statement_aux
                        FROM table_name
                        option_where
                        option_group
                        option_order
                        option_partition
                        option_limit
```

```
                        option_allow
option_json_distinct ::= JSON | DISTINCT | ε
select_statement_aux ::= select_clause | *
select_clause ::= selector option_as_idf ( , selector option_as_idf )*
selector ::= column_name | term | function_name ( option_selector )
             | CAST ( selector AS cql_type )
             | COUNT ( * )
option_selector ::= selector ( , selector )* | ε
option_as_idf ::= AS identifier | ε
option_where ::= WHERE where_clause | ε
option_group ::= GROUP BY group_by_clause | ε
option_order ::= ORDER BY ordering_clause | ε
option_partition ::= PER PARTITION LIMIT option_type_aux | ε
option_limit ::= LIMIT option_type_aux | ε
option_allow ::= ALLOW FILTERING | ε
option_type_aux ::= integer | bind_marker
bind_marker ::= ? | : identifier
where_clause ::= relation ( AND relation )*
relation ::= column_name operator term
           | ( column_name ( , column_name )* ) operator tuple_literal
           | TOKEN ( column_name ( , column_name )* ) operator term
operator ::= = | < | > | <= | >= | != | CONTAINS | CONTAINS KEY
tuple_literal ::= ( term ( , term )* )
group_by_clause ::= column_name ( , column_name )*
ordering_clause ::= column_name option_order (, column_name
                                                    option_order )*
option_order ::= ASC | DESC | ε
```

insert :

```
insert_statement ::= INSERT INTO table_name insert_statement_aux
                     if_not_exists   option_parameter
insert_statement_aux ::= names_values | json_clause
option_parameter ::= USING update_para ( AND update_para )* | ε
names_values ::= names VALUES tuple_literal
json_clause ::= JSON string option_default
option_default ::= DEFAULT default_aux | ε
default_aux ::= NULL | UNSET
names ::= ( column_name ( , column_name )* )
update_para ::= update_para_aux   option_type_aux
update_para_aux ::= TIMESTAMP | TTL
```

update:

```
update_statement ::= UPDATE table_name option_using
                     SET assignment ( , assignment )*
                     WHERE where_clause
                     option_if
assignment ::= column_name assignment_aux
assignment_aux ::= column_selection = term
                 | = aff_column_name
aff_column_name ::= column_name sign_aux term
                  | list_literal + column_name
sign_aux ::= + | -
column_selection ::= [ term ]   | . field_name   |   ε
option_if ::= IF option_if_aux   |   ε
option_if_aux ::= EXISTS | condition ( AND condition )*
condition ::= simple_selection operator term
simple_selection ::= column_name   column_selection
```

delete :

```
delete_statement ::= DELETE option_selection FROM table_name
                     option_using   WHERE   where_clause   option_if
```

batch :

```
batch_statement ::= BEGIN option_batch BATCH
                    option_using
                    modification_statement ( ; modification_statement )*
                    APPLY BATCH
option_batch ::= UNLOGGED | COUNTER | ε
option_using ::= USING update_para ( AND update_para )* | ε
modification_statement ::= insert_statement
                         | update_statement
                         | delete_statement
```

grant :

```
grant_statement ::= GRANT grant_role_statement
                                | grant_permission_statement
grant_role_statement ::= role_name TO role_name
grant_permission_statement ::= permissions ON resource TO role_name
```

```
permissions ::= ALL permissions_statement
                | permission permission_statement
permissions_statement ::= PERMISSIONS | ε
permission_statement ::= PERMISSION | ε
permission ::= CREATE | ALTER | DROP | SELECT | MODIFY
                | AUTHORIZE | DESCRIBE | EXECUTE
resource ::= ALL all_statement
             | KEYSPACE keyspace_name
             | table_statement table_name
             | ROLE role_name
             | FUNCTION function_name ( function_argument )
             | ( MBEAN | MBEANS ) string

function_argument ::= cql_type ( , cql_type )* | ε
all_statement ::= KEYSPACES | ROLES | MBEANS
                  | FUNCTIONS all_functions_statement
all_functions_statement ::= IN KEYSPACE keyspace_name | ε
table_statement ::= TABLE | ε

revoke :
revoke_statement ::= REVOKE revoke_statement_aux
revoke_statement_aux ::= revoke_role_statement
                         | revoke_permission_statement
revoke_role_statement ::= role_name FROM role_name
revoke_permission_statement ::= permissions ON resource FROM role_name

list :

list_statement ::= LIST list_statement_aux
list_statement_aux ::= list_role_statement | USERS
                       | list_permissions_statement
list_role_statement ::= ROLES of_role_option   no_recursive_option
of_role_option ::= OF role_name | ε
no_recursive_option ::= NORECURSIVE | ε
list_permissions_statement ::= permissions on_resource_option   of_option
on_resource_option ::= ON resource | ε
of_option ::= OF role_name no_recursive_option | ε

cassandra data types:

cql_type ::= native_type | collection_type | user_defined_type
| tuple_type | custom_type

native_type ::= ASCII | BIGINT | BLOB | BOOLEAN | COUNTER | DATE
```

```
                    | DECIMAL | DOUBLE | FLOAT | INET | INT | SMALLINT | TEXT
                    | TIME | TIMESTAMP | TIMEUUID | TINYINT | UUID | VARCHAR
                    | VARINT
collection_type ::= MAP < cql_type , cql_type >
                      | SET < cql_type >
                      | LIST < cql_type >
user_defined_type ::= udt_name
udt_name ::= option_keyspace identifier
option_keyspace ::= keyspace_name. | ε
tuple_type ::= TUPLE < cql_type ( , cql_type )* >
custom_type ::= string

collection_literal ::= { collection_aux } | list_literal
collection_aux ::= term    map_set_aux | ε
map_set_aux ::= map_literal_aux | set_literal_aux
map_literal_aux ::= term : term ( , term : term )*
set_literal_aux ::= ( , term )*
list_literal ::= [ list_literal_aux ]
list_literal_aux ::= term ( , term )* | ε
```