OQL BNF © ODMG 3.0

queryProgram	::=	*declaration* { ; *declaration*} [; *query*] | *query*
declaration	::=	*import* | *defineQuery* | *undefineQuery*
import	::=	**import** *qualifiedName* [**as** *identifier*]

defineQuery ::= **define** [**query**] *identifier* [**(** [*parameterList*] **)**]
as *query*

parameterList ::= *type identifier* { , *type identifier* }

undefineQuery ::= **undefine** [**query**] *identifier*

qualifiedName ::= *identifier* { . *identifier* }

query ::= *selectExpr*
| *expr*

selectExpr ::= **select** [**distinct**] *projectionAttributes*
fromClause
[*whereClause*]
[*groupClause*]
[*orderClause*]

projectionAttributes ::= *projectionList*
| *

projectionList ::= *projection* { , *projection* }

projection ::= *field*
| *expr* [**as** *identifier*]

fromClause ::= **from** *iteratorDef* { , *iteratorDef* }

iteratorDef ::= *expr* [[**as**] *identifier*]
| *identifier* **in** *expr*

whereClause ::= **where** *expr*

groupClause ::= **group by** *fieldList* [*havingClause*]

havingClause ::= **having** *expr*

orderClause ::= **order by** *sortCriteria*

sortCriteria ::= *sortCriterion* { , *sortCriterion* }

sortCriterion ::= *expr* [(**asc** | **desc**)]

expr ::= *castExpr*

castExpr ::= *orExpr*
| **(** *type* **)** *castExpr*

orExpr ::= *orelseExpr* { **or** *orelseExpr* }

orelseExpr ::= *andExpr* { **orelse** *andExpr* }

andExpr ::= *quantifierExpr* { **and** *quantifierExpr* }

quantifierExpr ::= *andthenExpr*
| **for all** *inClause* **:** *andthenExpr*
| **exists** *inClause* **:** *andthenExpr*

inClause ::= *identifier* **in** *expr*

andthenExpr ::= *equalityExpr* { **andthen** *equalityExpr* }

equalityExpr ::= *relationalExpr*
{ (= | !=) [*compositePredicate*] *relationalExpr* }
| *relationalExpr* { **like** *relationalExpr* }

relationalExpr ::= *additiveExpr*
{ (< | <= | > | >=) [*compositePredicate*]
additiveExpr }

compositePredicate ::= **some** | **any** | **all**

additiveExpr ::= *multiplicativeExpr* { + *multiplicativeExpr* }
| *multiplicativeExpr* { - *multiplicativeExpr* }
| *multiplicativeExpr* { **union** *multiplicativeExpr* }
| *multiplicativeExpr* { **except** *multiplicativeExpr* }
| *multiplicativeExpr* { || *multiplicativeExpr* }

multiplicativeExpr ::= *inExpr* { * *inExpr* }
| *inExpr* { / *inExpr* }
| *inExpr* { **mod** *inExpr* }
| *inExpr* { ***intersect*** *inExpr* }

inExpr ::= *unaryExpr* { **in** *unaryExpr* }

unaryExpr ::= **+** *unaryExpr*
| **-** *unaryExpr*
| **abs** *unaryExpr*
| **not** *unaryExpr*
| *postfixExpr*

postfixExpr ::= *primaryExpr* { **[** *index* **]** }
| *primaryExpr* { (**.** | **->**) *identifier* [*argList*] }

index ::= *expr* { **,** *expr*}
| *expr* **:** *expr*

argList ::= **(** [*valueList*] **)**

primaryExpr ::= *conversionExpr*
| *collectionExpr*
| *aggregateExpr*
| *undefinedExpr*
| *objectConstruction*
| *structConstruction*
| *collectionConstruction*
| *identifier* [*argList*]
| *queryParam*
| *literal*
| **(** *query* **)**

conversionExpr ::= **listtoset (** *query* **)**
| **element (** *query* **)**
| **distinct (** *query* **)**
| **flatten (** *query* **)**

collectionExpr ::= **first (** *query* **)**
| **last (** *query* **)**
| **unique (** *query* **)**
| **exists (** *query* **)**

aggregateExpr ::= **sum (** *query* **)**
| **min (** *query* **)**
| **max (** *query* **)**
| **avg (** *query* **)**
| **count (** (*query* | *****) **)**

```
undefinedExpr       ::= is_undefined ( query )
                      | is_defined ( query )

objectConstruction  ::= identifier ( fieldList )

structConstruction  ::= struct ( fieldList )

fieldList           ::= field { , field }

field               ::= identifier : expr

collectionConstruction ::= array ( [ valueList ] )
                         | set ( [ valueList ] )
                         | bag ( [ valueList ] )
                         | list ( [ valueList ] )
                         | list ( listRange )

valueList           ::= expr { , expr }

listRange           ::= expr .. expr

queryParam          ::= $ longLiteral

type                ::= [ unsigned ] short
                      | [ unsigned ] long
                      | long long
                      | float
                      | double
                      | char
                      | string
                      | boolean
                      | octet
                      | enum [ identifier . ] identifier
                      | date
                      | time
                      | interval
                      | timestamp
                      | set < type >
                      | bag < type >
                      | list < type >
                      | array < type >
                      | dictionary < type , type >
                      | identifier
```

identifier ::= *letter { letter | digit | _ }*

literal ::= *booleanLiteral*
| *longLiteral*
| *doubleLiteral*
| *charLiteral*
| *stringLiteral*
| *dateLiteral*
| *timeLiteral*
| *timestampLiteral*
| **nil**
| **undefined**

booleanLiteral ::= **true**
| **false**

longLiteral ::= *digit { digit }*

doubleLiteral ::= *digit { digit }* **.** *digit { digit }*
[(**E** | **e** *) [* **+** | **-** *] digit { digit }]*

charLiteral ::= **'** *character* **'**

stringLiteral ::= **"** *{ character }* **"**

dateLiteral ::= **date '** *longLiteral* **-** *longLiteral* **-** *longLiteral* **'**

timeLiteral ::= **time '** *longLiteral* **:** *longLiteral* **:** *floatLiteral* **'**

timestampLiteral ::= **timestamp '** *longLiteral* **-** *longLiteral* **-** *longLiteral*
longLiteral **:** *longLiteral* **:** *floatLiteral* **'**

character ::= *letter*
| *digit*
| *special-character*

letter ::= **A** | **B** | ... | **Z** |
a | **b** | ... | **z**

digit ::= **0** | **1** | ... | **9**

special-character ::= **?** | **_** | ***** | **%** | ****