Object Query Language Reference

Version: Itop 1.0

Overview

OQL aims at defining a subset of the data in a natural language, while hiding the complexity of the data model and benefit of the power of the object model (encapsulation, inheritance).

Its syntax sticks to the syntax of SQL, and its grammar is a subset of SQL.

As of now, only SELECT statements have been implemented.
Such a statement do return objects of the expected class. The result will be used by programmatic means (to develop an API like ITOp).

A famous example: the library

Starter

```
SELECT
    Book
```

Do return any book existing in the Database. No need to specify the expected columns as we would do in a SQL SELECT clause: OQL do return plain objects.

Join classes together

I would like to list all books written by someone whose name starts with 'Camus'

```
SELECT
    Book
    JOIN Artist ON Book.written_by = Artist.id
    WHERE Artist.name LIKE 'Camus%'
```

Note that there is no need to specify wether the JOIN is an INNER JOIN, or LEFT JOIN. This is well-known in the data model. The OQL engine will in turn create a SQL queries based on the relevant option, but we do not want to care about it, do we?

Now, you may consider that the name of the author of a book is of importance. This is the case if should be displayed anytime you will list a set of books, or if it is an important key to search for.
Then you have the option to change the data model, and define the name of the author as an *external field*. Such an external field would be defined by the external key **written_by** and the target attribute **name**. Let's qualify it has **writer_name**.
The query could then be simplified to:

```
SELECT
    Book
    WHERE Book.writer_name LIKE 'Camus%'
```

The Join occurs, but is 100% transparent to the OQL. It will happen each and every time those objects are queried so that the attribute **writer_name** becomes part of the properties of a book –though it will be read-only.

Inheritance

Now, as this is a modern library, several types of media are available: Audio, Video, Book. All of them have been declared as **Item** in the data model.

Let's list items not being produced by a French company:

```
SELECT
    Item
    JOIN Producer ON Item.produced_by = Producer.id
    WHERE Producer.country != 'France'
```

This query will return books as well, because a Book is an Item… that's due to classes inheritance: a Book inherits from Item.

OQL Statement

There is currently one single type of statement: SELECT

```
SELECT
    class_reference
    [class_joined]
    [WHERE expression]
```

- Note the absence of FROM clause, because OQL is aimed at returning objects, not values.

- *class_reference* indicates the class of objects that you want to select.
- *class_joined* indicates a series of classes that you want to join, in order to restrict the set of selected objects (remember, it makes no sense to mention columns).
- *where_condition* is an expression, very close to what could be found in an SQL SELECT statement.

class_reference

```
class_name [AS class_alias]
```

- *class_name* indicates the class of objects that you want to select.
- *class_alias* indicates an alias that will be used to refer to the given class, in the expressions found into the WHERE clause.

class_name or class_alias

```
name | `name`
```

Backticks must be used in the following circumstances:

- the name of the class is in conflict with a reserved word (example: JOIN…),
- the name of the class contains undesirable characters.

class_joined

```
JOIN class_reference
    ON class_left.external_key = class_right.id
```

- *class_reference* refers either to the class on the left of the join or the right… depending on the data model and the given external key.
- *class_right.id* has to be specified though it cannot be something else: it refers to the object that is pointed to by the other one. Class_right is an alias if any has been given.
- *class_left.external_key* indicates which attribute from which class should be pointing to class_right.id. In most cases, the external key attributes could be guessed, but the reference has to be specified explicitely anyway.

expression

```
literal
 | function
 | attribute
 | expression operator expression
 | (expression)
```

- *literal* is either a string (single or double quotes) or a number (only integers are supported).

- *function* is one of the verbs listed above, the arguments are a coma separated list of expressions
- *attribute* is a reference to an object property as defined in the data model, in the form *class_ref.attribute_code* – use of backticks is necessary to solve conflict with reserverd words or white chars.
- *operator* is any of binary operators listed below.
- expressions may be enclosed in parenthesis to cope with operators precedence.

Name	Description	Notes
AND	Logical AND	Synonym && not available
/	Division operator	
=	Equal operator	
>=	Greater than or equal operator	
>	Greater than operator	
<=	Less than or equal operator	
<	Less than operator	
LIKE	Simple pattern matching	
-	Minus operator	
!=, <>	Not equal operator	Synonym <> not available
NOT LIKE	Negation of simple pattern matching	
+	Addition operator	
*	Times operator	
-	Change the sign of the argument	
OR	Logical OR	Synonym || not available
IN	Check whether a value is within a set of values	
NOT INT	Check whether a value is not within a set of values	

function

```
verb(expression[,expression [,expression...]...] ...])
```

- *verb* is one the known function listed below

All of them are actually mapped to their equivalent in SQL. In other words, the same functions will be used in the resulting SQL queries that will be finally executed.

Therefore, the specification of those functions (number and type of arguments, returned values) stick to the specification of the underlying database.
Any limitation or side-effect, will be related to the version of the database engine.

The hyperlinks provided hereafter will direct you to the reference documentation of mySQL 5.0, which is the standard recommended database engine (used for qualification of the OQL processor).

Notes:

- Names are case-sensitive. They have to be uppercase in our implementation, though mySQL is less restrictive.
- So far, no synomym has been implemented (we kept one single name for a given function ; example: OQL implements DAY whereas mySQL implements DAY and DAYOFMONTH as an alias to the same function)

Function name	Description	Examples
IF	If/else construct	`IF(a=b, 'equals', 'differs')`
ELT	Return string at index number	`ELT(index, 'string1', 'string2', 'string3')`
COALESCE	Return the first non-NULL argument	
CONCAT	Return concatenated string	`CONCAT(firstname, ' ', lastname)`
SUBSTR	Return the substring as specified	`SUBSTR('abcdef', 2, 3)`
TRIM	Remove leading and trailing spaces	`TRIM('  blah  ')`
DATE	Extract the date part of a date or datetime expression	`DATE()`
DATE_FORMAT	Format date as specified	`DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y')`
CURRENT_DATE	Return the current date	`CURRENT_DATE()`
NOW	Return the current date and time	`NOW()`
TIME	Extract the time portion of the expression passed	`TIME()`
TO_DAYS	Return the date argument	`TO_DAYS('2009-05-01')`

Function name	Description	Examples
	converted to days	
FROM_DAYS	Convert a day number to a date	`FROM_DAYS(12345)`
YEAR	Return the year from the date passed	`YEAR(DATE())`
MONTH	Return the month from the date passed	`MONTH(DATE())`
DAY	Return the day of the month (0-31)	`DAY(DATE())`
DATE_ADD	Add time values (intervals) to a date value See allowed interval units below	`DATE_ADD(NOW() INTERVAL 1 HOUR)`
DATE_SUB	Substract time values (intervals) from a date value See allowed interval units below	`DATE_SUB(NOW() INTERVAL 5 MINUTE)`
ROUND	Round the argument	`ROUND(12.356, 2)`
FLOOR	Return the largest integer value not greater than the argument	`FLOOR(12.356)`
INET_ATON	Return the numeric value of an IP address	`INET_ATON('15.15.121.12')`
INET_NTOA	Return the IP address from a numeric value	`INET_NTOA(1231654)`

The list of time interval units currently supported by the functions DATE_ADD and DATE_SUB, is a subset of the values allowed in mySQL.

OQL does accept:

Time interval units
YEAR
MONTH
DAY
HOUR
MINUTE
SECOND

BNF Grammar

```
select-query ::= SELECT class-reference [class-joined] [WHERE expression]
class-reference ::= name [AS name]
class-joined ::= JOIN class-reference ON name.name = name.id
name ::= string | `string`
expression ::=
        scalar
        | expression operator expression
        | (expression)
scalar ::= number | ‘string’ | “string” | column
operator ::= AND | OR | = | <> | != | > | >= | < | <= | LIKE | NOT LIKE
column ::= name | name.name
```