Xaraya
Request for Comments: 0053
Category: Standards Track

M.L. Lutolf
Xaraya Development Group
September 2004

Structured Queries in Xaraya

Status of this Memo

This document specifies an Xaraya standards track protocol for the Xaraya community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Xaraya Official Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This RFC describes an implementation of structured queries through query abstraction, based on the ideas presented in rfc35[1].

[1] http://www.xaraya.com/documentation/rfcs/rfc0035.html

Table of Contents

1 Introduction....4

2 Objectives and Limitations....6

3 Description....7

4 The xarQuery API....8

4.1 Query Creation....8
4.1.1 xarQuery([type[,table[,array([field1],[field2]..)]]])....8
4.1.2 open([connection])....9
4.1.3 close([connection])....9
4.1.4 openconnection([connection])....9

4.2 Tables and Fields....10
4.2.1 addfield(name[,value]) or addfield(fieldarray)....10
4.2.2 addfields(array([field1],[field2]..))....10
4.2.3 addtable(name[,alias]) or addtable(tablearray)....11
4.2.4 addtables(array([table1],[table2]..))....11
4.2.5 getfield(fieldname)....12
4.2.6 removefield(fieldname)....12
4.2.7 join(fieldname1,fieldname2)....13
4.2.8 settable(name[,alias]) or settable(tablearray)....13

4.3 Conditions....13
4.3.1 eq(fieldname,value)....13
4.3.2 ge(fieldname,value)....14
4.3.3 gt(fieldname,value)....14
4.3.4 in(fieldname,valuearray)....14
4.3.5 le(fieldname,value)....15
4.3.6 like(fieldname,value)....15
4.3.7 lt(fieldname,value)....16
4.3.8 ne(fieldname,value)....16

4.4 Execution and output....17
4.4.1 output()....17
4.4.2 qecho()....17
4.4.3 row([rowno])....18
4.4.4 run([statement][,flag])....18
4.4.5 tostring()....19

4.5 Miscellaneous....20
4.5.1 addorder(fieldname[,direction])....20
4.5.2 clearconditions()....20
4.5.3 clearfields(fieldname)....20
4.5.4 clearfields()....21
4.5.5 getrows()....21
4.5.6 getrowstodo()....22
4.5.7 getstartat()....22
4.5.8 gettype()....23
4.5.9 nolimits()....23
4.5.10 setorder(fieldname[,direction])....23
4.5.11 setrowstodo(rowno)....24
4.5.12 setstartat(rowno)....24
4.5.13 settype(type)....25
4.5.14 uselimits()....25

4.6 Examples....25

Author's Address **29**

A Example appendix **30**

Intellectual Property and Copyright Statements **31**

1. Introduction

Structured queries are a way of making the Query Abastraction has been made necessary to give us the hability to generate efficient SQL queries which are portable across all our Database Pool. It will provide us a way to create queries without any preocupations about the underlying storage system.

Typical life cycle of a query:

```
    Assembly Phase:

        - define the type of query (SELECT, INSERT, etc.)
        - define the tables
        - define the fields involved
        - define conditions and filters
        - how is the data ordered and presented (for SELECTS)

    ------------------------------------------------------------------------

    Execution Phase:

        - run the query              communicate with the database

    ------------------------------------------------------------------------

    Output Phase:

        - collect the output
        - rework as necessary (add field names, do security checks, etc.)
        - send to output templates or
        - send to another application for further processing
```

[N.B.: Xaraya uses adodb to communicate with the underlying database(s). The structured query does this at one specific point (class method), making it relatively simple to adjust should the database layer change.]

Structured queries are essentially a convenience mechanism, which allows Xaraya to work with database queries in a more flexible and dynamic way in a number of areas:

Defaults

- Standard database connections are automatically handled as required.
- If field alias names are used, these are automatically returned in the output.
- If no fields of a table are explicitly requested, all the fields of the table are returned in the output with the field names as defined in the database table.
- For queries using LIMIT, the total number of records without the LIMIT clause can be recalled in the output.
- Binding variables are automatically used in queries for greater performance under some databases.
- Enhancements to query manipulation, inasmuch as they don't come adodb itself, can be made through a single point, rather than having to go through the entire codebase and change each single query.

Dynamic Queries

- Records to be retrieved and starting point can be easily specified for queries that show only a subset of records on a single page (pager mechanism).
- Using simple if-statements fields and conditions can be easily added or removed. This is useful in pages that create reports based on a numer of alternative criteria. The members list page in the roles module is one example of this.
- Query types can be changed on the fly. As an example, using structured queries in object creation in the roles module, you can start with a SELECT query to check if the object already exists, then turn the same query into an INSERT or UPDATE depending on the result.

Storage and Retrieval

- Structured queries can be easily passed from one page to the next, and modified on the fly. One example is the messaging page in the roles module, which can be passed a query (of a set of users) from the users view page, but will then modify that query according to new user input.
- Queries can be serialized and stored for later retrieval, for instance in a session variable. This makes it possible for a user to maintain certain "current items" as he/she moves through a site. In a report-heavy module such as an accounting package, a user can define his/her own custom reports that can be easily modified on the fly.

Query Algebra

- Having structured queries that can be passed as variables and manipulated through a well defined API opens the door to combining such queries in various ways. A "query algebra" can be defined on the basis of set operations (intersection, union, difference etc.). .
- One simple application of this would be to represent INSERTS, UPDATES and DELETES through a union result of combining several structured queries. Disparate tables can then be updated by applying the result query in one command.
- Another possible application is the creation of generalized DD-like structures that allow for adding extra data properties, say in the roles tree, that depend on the instance in question and are not the same for all roles.

2. Objectives and Limitations

The objective in implementing structured queries is to make query manipulation more flexible and dynamic, while not making the API to do so more complicated than what is currently used, i.e. concatenated strings.

This means we are not interested in creating a complete query manipulation language per se. The implementation currently does not support CREATE clauses for instzance, although it can be extended to do so

3. Description

Structured queries are implemented using the *xarQuery* class. The implementation chosen purposely works with only this single class, which represents a structured query object. All manipulation of the query is performed using this class's methods and properties.

A simple example of a structured query:

```
    Assembly Phase:

        $q = new xarQuery([type],[table],[fields]);     Set the query up

        $q->addtable(...);                              Add more tables

        $q->addfield(...);                              Add more fields

        $q->eq(...);                                    Add an EQ clause

    ------------------------------------------------------------------------

    Execution Phase:

        $q->run();                                      Run the query

           or

        if (!$q->run()) return;                         Run and return an
exception of failed

    ------------------------------------------------------------------------

    Output Phase:

        $data['rows'] = $q->output();                   Send the output to a
template

           or

        $myobject = new Object($q->output());           Use the output to create
an object

           or

        foreach ($q->output() as $key => $value) {      Use the output in a loop
            ....
        }
```

- The first line of any structured query must be new xarQuery() in order for a query to be created.
- By convention the query object is called $q ($q1, $q2..if there are more), but any name that PHP will accept as an object name is possible.
- Other than the first line, the lines in the assemply phase do not need to be in any particular order.
- The $q->run(); line creates the final query statement and sends it to the database. The query object then receives the results of the query. In the case of SELECTS this is the recordset returned from the database.
- The expression $q->output(); creates an associative 2-dimensional array of the records found, where in each row the key of an element is the field name and the value is the contents of the field. An empty result is reurned as an empty array..

4. The xarQuery API

Example: Recalling a user's data from the database. The data of the user with the uname $this->uname is returned as an array.

```
$q = new xarQuery('SELECT',$this->rolestable);
$q->eq('xar_uname',$this->uname);
if (!$q->run()) return;
$foo = $q->output();
```

The syntax of the class call is

```
$q = new xarQuery('SELECT'|'INSERT'|'UPDATE'|'DELETE',
                  {array of query tables},
                  {array of fields}
);
```

The class methods (to be completed):

4.1 Query Creation

4.1.1 xarQuery([type[,table[,array([field1],[field2]..)]]])

With the constructor method xarQuery() we can create a new structured query object that subsequently be elaborated on by the objects methods.

In general the constructor will include only the type parameter, which defines the type of query created (SELECT, UPDATE etc.). The table and fields parameters are used as a shorthand for simple SELECT queries.

4.1.1.1 Parameters

type

String whose value can be SELECT, UPDATE, INSERT or DELETE. The default is SELECT.

table

String representing a database table name.

array([field1],[field2]..)

An array of database fields as strings or arrays. If [fieldx] is a string, it must correspond to a database field that is uniquely identified, such as "mytable.foo1".

If [fieldx] is an array, it can have the form described in the addfields() method.

Note that in the case of a single database field, this parameter can also be a string representing the field's name.

4.1.1.2 Syntax examples

```
$q = new xarQuery();                      // creates an empty query object

$q = new xarQuery('INSERT');              // creates a query object of type INSERT

$q = new xarQuery('SELECT','xar_roles');  // creates a query object on the roles
table

$q = new xarQuery('SELECT','xar_roles','xar_name');
                                          // adds a single field to the
constructor

$q = new xarQuery('SELECT','xar_roles',array('xar_uid','xar_name'));
```

```
                                                // adds a two fields to the constructor
```

A longer way of writing the last example would be:

```
$q = new xarQuery('SELECT');
$q->addtable('xar_roles');
$q->addfield('xar_uid');
$q->addfield('xar_name');
```

4.1.2 open([connection])

The open() method opens a database connection. In adodb it is equivalent to

```
$this->dbconn = xarDBGetConn();
```

The database connection is created in the xarQuery() constructor method. This method is used after unserializing a stored query and before running it.

4.1.2.1 Parameters

connection (to be implemented)

The connection to be opened. If the connection already exists it is opened. If it does not exist it is created.

4.1.2.2 Syntax examples

```
$q = unserialize(xarSessionGetVar('foo'));
$q->open();
$q->run();
```

4.1.3 close([connection])

The close() method closes a database connection.

4.1.3.1 Parameters

connection (to be implemented)

The connection to be closed

4.1.3.2 Syntax examples

```
$q->close();
```

4.1.4 openconnection([connection])

The openconnection() method closes a database connection.

4.1.4.1 Parameters

connection (to be implemented)

The connection to be closed

4.1.4.2 Syntax examples

```
$q->close();
```

4.2 Tables and Fields

4.2.1 addfield(name[,value]) or addfield(fieldarray)

The addfield() method adds a single field to the current query. This method is used as a shorthand for adding fields to a SELECT query. For SELECT queries it is also possible to use the addfields() method.

4.2.1.1 Parameters

name

A string representing the name of a database field to be added to the query. The parameter can have the form "foo AS bar", where bar is the *alias* of the field foo. If the alias is given it will be used when generating output with the output() method. Only SELECT queries use the alias.

value

The value to be assigned to the field. The value must correspond to the field type in the database, or to a type that PHP can convert to the field type. The value is used for UPDATE and INSERT queries. Queries of the type SELECT or DELETE ignore it

fieldarray

An array representing the database field to be added to the query. It has the general form:

```
array('name'=>'mytable.foo1'[,'value'=>'bar'[,'alias'=>'foobar']])
```

As noted above, the value is only used by INSERTS and UPDATES, while the alias is only used by SELECTS.

4.2.1.2 Syntax examples

```
$q = new xarQuery('INSERT','xar_roles');
$q->addfield('xar_name','Sharon Stone');
$q->addfield('xar_uname','moviestar');
$q->qecho();

displays

INSERT INTO xar_roles (xar_name, xar_uname) VALUES ('Sharon Stone', 'moviestar')

$q = new xarQuery('SELECT','xar_roles');
$q->addfield('xar_uname');
$q->eq('xar_name','Sharon Stone');
$q->run();

$row = $q->row();
echo $row['uname'];

displays

moviestar
```

4.2.2 addfields(array([field1],[field2]..))

The addfields() method adds an array of fields to the current query. This method is used as a shorthand for adding fields to a SELECT query. For other query types it is better to use the addfield() method.

4.2.2.1 Parameters

array([field1],[field2]..)

An array of database fields as strings or arrays. If [fieldx] is a string, it must correspond to a database field that is uniquely identified, such as "mytable.foo1".

If [fieldx] is an array, it can have the form

```
array('name'=>'mytable.foo1'[,'value'=>'bar'[,'alias'=>'foobar']])
```

See the addfield() method for details.

4.2.2.2 Syntax examples

```
$q = new xarQuery('SELECT','mytable');
$q->addfields(array('foo1','foo2','foo3'));
$q->run();
```

4.2.3 addtable(name[,alias]) or addtable(tablearray)

The addtable() method adds a single table to the current query. This method is used as a shorthand for adding tables to a SELECT query. For a shorthand notation to add tables to a SELECT query, you can also use the addtables() method.

4.2.3.1 Parameters

name

A string representing the name of a database table. The name can have the form "foo bar" where bar is the *alias* of foo. Aliases are used to help differentiate between fields of the same name in different tables.

alias

A string representing the alias of a database table.

tablearray

An array representing the name of a database table. The array has the form:

```
array('name'=>'mytable'[,'alias'=>'tablealias'])
```

The name and alias correspond to the form "foo bar" given in the previous point.

4.2.3.2 Syntax examples

The following example selects the uids of all the groups the user Sharon Stone is a member of. Note the alternate syntaxes in the two addtable() methods.

```
$q = new xarQuery('SELECT',);
$q->addfield('r.xar_parentid');
$q->addtable('xar_roles','r');
$q->addtable('xar_rolemembers rm');
$q->join('r.xar_uid','rm.xar_uid');
$q->eq('xar_name','Sharon Stone');
$q->qecho();

displays

SELECT r.xar_parentid FROM xar_roles r, xar_rolemembers rm
WHERE r.xar_uid = rm.xar_uid AND xar_name = 'Sharon Stone'
```

4.2.4 addtables(array([table1],[table2]..))

The addtables() method adds an array of tables to the current query. This method is used as a shorthand for adding tables to a SELECT query. For other query types, and also for more complicated SELECTs, it is better to use the addtable() method.

4.2.4.1 Parameters

array([table1],[table2]..)

An array of database tables as strings or an array. If [tablex] is a string, it must correspond to a database table.

If [tablex] is an array, it can have the form

```
array('name'=>'mytable'[,'alias'=>'tablealias'])
```

See the addtable() method for details.

4.2.4.2 Syntax examples

```
...
$q->addtables(array('table1','table2','table3'));
...
```

4.2.5 getfield(fieldname)

The getfield() method returns a field in a query or NULL if the field does not exist.

4.2.5.1 Parameters

fieldname

A string representing the name or alias of a field in the query.

4.2.5.2 Syntax examples

```
...
$q->getfield('foo');
...
```

4.2.6 removefield(fieldname)

The removefield() method removes a field from a query or does nothing if the field does not exist.

4.2.6.1 Parameters

fieldname

A string representing the name or alias of a field in the query.

4.2.6.2 Syntax examples

```
...
$q->removefield('foo');
...
```

4.2.7 join(fieldname1,fieldname2)

The join() method removes a field from a query or does nothing if the field does not exist.

4.2.7.1 Parameters

fieldname1, fieldname2

Strings representing the name or alias of fields in two tables in the query that will be joined.

4.2.7.2 Syntax examples

The following example selects the uids of all the groups the user Sharon Stone is a member of:

```
$q = new xarQuery('SELECT',);
$q->addfield('r.xar_parentid');
$q->addtable('xar_roles','r');
$q->addtable('xar_rolemembers rm');
$q->join('r.xar_uid','rm.xar_uid');
$q->eq('xar_name','Sharon Stone');
$q->qecho();

displays

SELECT r.xar_parentid FROM xar_roles r, xar_rolemembers rm
WHERE r.xar_uid = rm.xar_uid AND xar_name = 'Sharon Stone'
```

4.2.8 settable(name[,alias]) or settable(tablearray)

The settable() method is a convenience method that clears all the query's current tables and adds a single table. It is an application of the cleartables() method followed by the addtable() method. See the addtable() method for details on the syntax.

4.3 Conditions

4.3.1 eq(fieldname,value)

The eq() method adds an EQUALS condition to a query.

4.3.1.1 Parameters

fieldname

String representing the name or alias of a field.

value

The value to be assigned to the field as a condition. The value must correspond to the field type in the database, or to a type that PHP can convert to the field type.

4.3.1.2 Syntax examples

The following example returns a record from the roles table corresponding to the current user.

```
$q = new xarQuery('SELECT','xar_roles');
$q->eq('xar_uid', xarSessionGetVar('uid'));
$q->run();
```

The following example returns all records from the roles table that represent groups.

```
$q = new xarQuery('SELECT','xar_roles');
$q->eq('xar_type', 1);
$q->run();
```

4.3.2 ge(fieldname,value)

The ge() method adds a GREATER THAN condition to a query.

4.3.2.1 Parameters

fieldname

String representing the name or alias of a field.

value

The value to be assigned to the field as a condition. The value must correspond to the field type in the database, or to a type that PHP can convert to the field type.

4.3.2.2 Syntax examples

The following example returns all records from the privileges table that represent READ access or higher.

```
$q = new xarQuery('SELECT','xar_roles');
$q->ge('xar_level', 200);
$q->run();
```

4.3.3 gt(fieldname,value)

The gt() method adds a GREATER THAN condition to a query.

4.3.3.1 Parameters

fieldname

String representing the name or alias of a field.

value

The value to be assigned to the field as a condition. The value must correspond to the field type in the database, or to a type that PHP can convert to the field type.

4.3.3.2 Syntax examples

The following example returns all records from the privileges table that represent READ access or higher. Same result as the example in ge().

```
$q = new xarQuery('SELECT','xar_privileges');
$q->gt('xar_level', 100);
$q->run();
```

4.3.4 in(fieldname,valuearray)

The in() method adds an IN condition to a query.

4.3.4.1 Parameters

fieldname

String representing the name or alias of a field.

valuearray

An array of values to be compared to the field as a condition. The values must correspond to the field type in the database, or to a type that PHP can convert to the field type.

4.3.4.2 Syntax examples

The following example returns the records from the roles table corresponding to users that are not validated or pending approval:

```
$q = new xarQuery('SELECT','xar_roles');
$q->eq('xar_type', 0);
$q->in('xar_state', array(ROLES_STATE_NOTVALIDATED,ROLES_STATE_PENDING));
$q->run();
```

4.3.5 le(fieldname,value)

The le() method adds a LESS THAN OR EQUAL condition to a query.

4.3.5.1 Parameters

fieldname

String representing the name or alias of a field.

value

The value to be assigned to the field as a condition. The value must correspond to the field type in the database, or to a type that PHP can convert to the field type.

4.3.5.2 Syntax examples

The following example returns all records from the roles table that represent users that are deleted, inactive or not validated.

```
$q = new xarQuery('SELECT','xar_roles');
$q->le('xar_state', ROLES_STATE_NOTVALIDATED);
$q->run();
```

4.3.6 like(fieldname,value)

The like() method adds a LIKE condition to a query. This condition is used to create simple conditions based on comparisons for string fields.

4.3.6.1 Parameters

fieldname

String representing the name or alias of a field.

value

The value to be assigned to the field as a condition. The value must correspond to the field type in the database, or to a type that PHP can convert to the field type.

Adding the % character to the value turns it into a wild card, as shown in the example below.

4.3.6.2 Syntax examples

The following example returns all records from the roles table that represent roles whose name begin with "John".

```
$q = new xarQuery('SELECT','xar_roles');
$q->like('xar_name', 'John%');
$q->run();
```

4.3.7 lt(fieldname,value)

The lt() method adds a LESS THAN condition to a query.

4.3.7.1 Parameters

fieldname

String representing the name or alias of a field.

value

The value to be assigned to the field as a condition. The value must correspond to the field type in the database, or to a type that PHP can convert to the field type.

4.3.7.2 Syntax examples

The following example returns all records from the roles table with a uid less than 5.

```
$q = new xarQuery('SELECT','xar_roles');
$q->lt('xar_xaruid', 5);
$q->run();
```

4.3.8 ne(fieldname,value)

The ne() method adds an NOT EQUALS condition to a query.

4.3.8.1 Parameters

fieldname

String representing the name or alias of a field.

value

The value to be assigned to the field as a condition. The value must correspond to the field type in the database, or to a type that PHP can convert to the field type.

4.3.8.2 Syntax examples

The following example returns the records from the roles table corresponding to groups.

```
$q = new xarQuery('SELECT','xar_roles');
$q->ne('xar_type', 0);
$q->run();
```

- *regex(fieldname,value):* add a REGEX condition to the query.
- *qand(condition1,condition2):* link two conditions with an AND.

- *qor(condition1,condition2):* link two conditions with an OR.
- *regex(fieldname,value):* add a REGEX condition to the query.
- *getcondition($mycondition):* return a condition from the statement.
- *removecondition($mycondition):* remove a condition from the statement.

4.4 Execution and output

4.4.1 output()

The output() method returns the output of a query as a 2 dimensional array where the keys of row elements corespond to the field names in the database. If the fields were defined with aliases, the aliases are used as the keys for the row elements.

The output() method works with SELECT statements. If the query finds no records the output method will return an empty array. To retrieve a single output row use the row() method.

4.4.1.1 Parameters

none

4.4.1.2 Syntax examples

The following example selects and displays some data from the roles module.

```
$q = new xarQuery('SELECT','xar_roles','xar_name');
$q->lt('xar_uid',4);
$q->run();
echo var_dump($q->output());

displays:

array(3) {
  [0]=>array(1) {["xar_name"]=>string(9) "Everybody"}
  [1]=>array(1) {["xar_name"]=>string(9) "Anonymous"}
  [2]=>array(1) {["xar_name"]=>string(5) "Admin"}
}
```

4.4.2 qecho()

The qecho() method echos a query statement to the screen. This method is used for debugging queries. Note that the output of this method will vary according to whether the query has actually been run, as illustrated below.

4.4.2.1 Parameters

none

4.4.2.2 Syntax examples

The following example displays a SELECT statement that queries the xar_roles table.

```
$q = new xarQuery('SELECT','xar_roles');
$q->eq('xar_name','Anonymous');
$q->qecho();

displays:

SELECT * FROM xar_roles WHERE xar_name = 'Anonymous'
```

The same example displays lists the fields by name once the query has actually been run.

```
$q = new xarQuery('SELECT','xar_roles');
$q->eq('xar_name','Anonymous');
$q->run();
$q->qecho();

displays:

SELECT xar_uid, xar_name, xar_type, xar_users, xar_uname, xar_email, xar_pass,
xar_date_reg, xar_valcode, xar_state, xar_auth_module FROM xar_roles WHERE
xar_name = 'Anonymous'
```

The following example displays an UPDATE statement that updates the xar_roles table.

```
$q = new xarQuery('UPDATE','xar_roles');
$q->addfield('xar_state',ROLES_STATE_ACTIVE);
$q->eq('xar_name','Sharon Stone');
$q->qecho();

displays:

UPDATE xar_roles SET xar_state = 3 WHERE xar_name = 'Sharon Stone'
```

4.4.3 row([rowno])

The row() method returns a single output row of a query as a 1 dimensional array where the keys of row elements corespond to the field names in the database. If the fields were defined with aliases, the aliases are used as the keys for the row elements.

The row() method works with SELECT statements. If the query finds no records the output method will return an empty array. To retrieve more than one output row at a time use the output() method.

4.4.3.1 Parameters

rowno

An integer that refers to the row number of the output record to be retrieved. The default value is rowno = 0, or the first row in the output recordset.

Note that since the sequence of query output may be database dependent and the query itself may include sorting directives, in general nothing can be said about the sequence in which the output arrives. It is the responsibility of the programmer to ensure the rowno parameter is a valid number.

4.4.3.2 Syntax examples

The following example selects and displays some data of the Anonymous user in the roles data.

```
$q = new xarQuery('SELECT','xar_roles',array('xar_uid','xar_name'));
$q->eq('xar_name','Anonymous');
$q->run();
echo var_dump($q->row());

displays:

array(2) {
  ["xar_uid"]=>  string(1) "2"
  ["xar_name"]=>  string(9) "Anonymous"
}
```

4.4.4 run([statement][,flag])

The run() method assembles and executes a a query. If a string with a statement is given it is directly executed. If $flag is 1 then the method output() contains the resultset. If $flag is 0 then the public variable $result contains the adodb resultset.

4.4.4.1 Parameters

statement

String representing a valid SQL statement. If this parameter is not empty, the run method executes it. This is equivalent to

```
$this->dbconn->Execute($statement);
```

In this case the query output can be retrieved as $q->result. No output is available to the output() method.

flag

If this parameter is set to true or 1, the query output is available via the output() method.

If this parameter is set to false or 0, the query output can be gotten from the public variable $q->result.

The default value for this parameter is *true*.

4.4.4.2 Syntax examples

The following example inserts a record into the xar_rolemembers table by executing a simple SQL statement. The code below it is equivalent.

```
$q = new xarQuery();
$q->run("INSERT INTO xar_rolemembers (xar_uid, xar_parentid) VALUES (301, 300)");

$q = new xarQuery('INSERT','xar_rolemembers');
$q->addfield('xar_parentid',300);
$q->addfield('xar_uid',301);
$q->run();
```

The following example gets the record of the current user in the xar_roles table by executing a simple SQL statement. The code below it is equivalent.

```
$q = new xarQuery();
$q->run("SELECT * FROM xar_roles WHERE xar_uid = " . xarSessionGetVar('uid'));

$q = new xarQuery('SELECT','xar_roles');
$q->eq('xar_uid',xarSessionGetVar('uid'));
$q->run();
```

Note that in the second case xarQuery by default assumes all fields are to be included in the SELECT statement. Furthermore, although the SELECT executed is the same in both cases, running the raw statement means the output will be available as $q->result, whereas in the second case it will be available through the output() method.

4.4.5 tostring()

The tostring() method returns a query statement.

4.4.5.1 Parameters

none

4.4.5.2 Syntax examples

The following example returns a SELECT statement that queries the xar_roles table.

```
$q = new xarQuery('SELECT','xar_roles');
$q->eq('xar_name','Anonymous');
$q->tostring();

returns:

SELECT * FROM xar_roles WHERE xar_name = 'Anonymous'
```

4.5 Miscellaneous

4.5.1 addorder(fieldname[,direction])

The addorder() method adds a ORDER clause to a query. If more than one addorder() is applied, the multiple sorts are applied in the order they are enocuntered. If no prior sort exists, addorder behaves like the setorder() method.

4.5.1.1 Parameters

fieldname

String representing the name or alias of a field.

direction

String representing the direction of the sort. Allowed values are ASC (ascending) and DESC (descending). The default value is ASC.

4.5.1.2 Syntax examples

```
$q = new xarQuery('SELECT','xar_security_levels','xar_leveltext');
$q->setrowstodo(3);
$q->addorder('xar_leveltext','DESC');
$q->run();
echo var_dump($q->output());

displays:

array(3) {
    [0]=>  array(1) { ["xar_leveltext"]=>  string(11) "ACCESS_READ" }
    [1]=>  array(1) { ["xar_leveltext"]=>  string(15) "ACCESS_OVERVIEW" }
    [2]=>  array(1) { ["xar_leveltext"]=>  string(11) "ACCESS_NONE" }
}
```

4.5.2 clearconditions()

The clearconditions() method removes all the conditions from a query.

4.5.2.1 Parameters

none

4.5.2.2 Syntax examples

```
...
$q->clearconditions();
...
```

4.5.3 clearfields(fieldname)

The clearfield() method removes a single field from a query.

4.5.3.1 Parameters

fieldname

String representing the name or alias of a field.

4.5.3.2 Syntax examples

```
$q = new xarQuery('SELECT','xar_roles');
$q->addfield('xar_uid');
$q->addfield('xar_name');
$q->eq('xar_name','Anonymous');
$q->qecho();

displays

SELECT xar_uid, xar_name FROM xar_roles WHERE xar_name = 'Anonymous'

$q->clearfield('xar_name');
$q->qecho();

displays

SELECT xar_uid FROM xar_roles WHERE xar_name = 'Anonymous'
```

4.5.4 clearfields()

The clearfields() method removes all the fields from a query.

4.5.4.1 Parameters

none

4.5.4.2 Syntax examples

```
$q = new xarQuery('SELECT','xar_roles');
$q->addfield('xar_uid');
$q->addfield('xar_name');
$q->eq('xar_name','Anonymous');
$q->qecho();

displays

SELECT xar_uid, xar_name FROM xar_roles WHERE xar_name = 'Anonymous'

$q->clearfields();
$q->qecho();

displays

SELECT * FROM xar_roles WHERE xar_name = 'Anonymous'
```

Note in the last example the fact that if all fields are cleared the query will default to *, i.e. *all* fields will be chosen.

4.5.5 getrows()

The getrows() method returns the total number of rows that a query of type SELECT will return, regardless of

whether a LIMIT clause has been defined. (see the setrowstodo() method.)

4.5.5.1 Parameters

none

4.5.5.2 Syntax examples

```
$q = new xarQuery('SELECT','xar_security_levels');
$q->run();
$q->getrows();

returns:

10
```

4.5.6 getrowstodo()

The getrows() method returns the number of rows that a query of type SELECT will return. This method only returns a non zero result if a LIMIT clause has been defined for the query. See also the setrowstodo() method.

4.5.6.1 Parameters

none

4.5.6.2 Syntax examples

```
$q = new xarQuery('SELECT','xar_security_levels');
$q->setstartat(3);
$q->setrowstodo(5);
$q->run();
$total1 = count($q->output());
$total2 = $q->getrowstodo();

returns:

$total1 = 5;     (records 3 to 8)
$total2 = 5;
```

4.5.7 getstartat()

The getstartat() method returns the row number query output begins at. This applies to SELECT statements with LIMIT clauses. A typical example is the input for a pager that tells it which rows are being displayed on a page. See also the setstartat() method.

4.5.7.1 Parameters

none

4.5.7.2 Syntax examples

```
$q = new xarQuery('SELECT','xar_roles');
$q->setstartat(100);
$q->setrowstodo(20);
echo $q->getstartat();

displays:
```

```
100
```

4.5.8 gettype()

The gettype() method returns the type of the query, e.g. SELECT, UPDATE etc. See also the settype() method.

4.5.8.1 Parameters

none

4.5.8.2 Syntax examples

```
$q = new xarQuery('SELECT','xar_roles');
$q->eq('xar_name','Anonymous');
echo $q->gettype();

displays:

SELECT
```

4.5.9 nolimits()

The nolimits() method disables the LIMIT clause in a SELECT statement, if such a clause is defined. The SELECT statement will return *all* the records found. By default limits are in effect unless disabled by this method. See also the uselimits() method.

4.5.9.1 Parameters

none

4.5.9.2 Syntax examples

```
$q = new xarQuery('SELECT','xar_security_levels');
$q->setstartat(3);
$q->setrowstodo(5);
$q->run();
$total1 = count($q->output());
$q->nolimits();
$q->run();
$total2 = count($q->output());

returns:

$total1 = 5;     (records 3 to 8)
$total2 = 10;    (all the records)
```

4.5.10 setorder(fieldname[,direction])

The setorder() method sets the ORDER clause to a query. If such a clause already exists it is replaced. See also the addorder() method.

4.5.10.1 Parameters

fieldname

String representing the name or alias of a field.

direction

String representing the direction of the sort. Allowed values are ASC (ascending) and DESC (descending). The default value is ASC.

4.5.10.2 Syntax examples

```
$q = new xarQuery('SELECT','xar_security_levels','xar_leveltext');
$q->setrowstodo(3);
$q->setorder('xar_leveltext','DESC');
$q->run();
echo var_dump($q->output());

displays:

array(3) {
    [0]=>  array(1) { ["xar_leveltext"]=>  string(11) "ACCESS_READ" }
    [1]=>  array(1) { ["xar_leveltext"]=>  string(15) "ACCESS_OVERVIEW" }
    [2]=>  array(1) { ["xar_leveltext"]=>  string(11) "ACCESS_NONE" }
}
```

4.5.11 setrowstodo(rowno)

The setrowstodo() method sets the number of rows that a query of type SELECT will return. This method, together with the setstartat() method, defines a LIMIT clause for the query. See also the getrowstodo() method.

4.5.11.1 Parameters

rowno

A number that defines how many rows the query will return. If the total number of rows the query selects is smaller than rowno, then the method has no effect.

4.5.11.2 Syntax examples

```
$q = new xarQuery('SELECT','xar_security_levels');
$q->setstartat(3);
$q->setrowstodo(5);
$q->run();
$total1 = count($q->output());

returns:

$total1 = 5;     (records 3 to 8)
```

4.5.12 setstartat(rowno)

The getstartat() method sets the row number query output begins at. This applies to SELECT statements with LIMIT clauses. A typical example is the input for a pager that tells it which rows are being displayed on a page. See also the getstartat() method.

4.5.12.1 Parameters

rowno

Number representing the row query input starts at. The first row is 1. It is the programmer's responsibility to ensure the rowno parameter has a reasonable value.

4.5.12.2 Syntax examples

```
$q = new xarQuery('SELECT','xar_roles');
$q->setstartat(100);
$q->setrowstodo(20);
echo $q->getstartat();

displays:

100
```

4.5.13 settype(type)

The gettype() method sets the type of the query, e.g. SELECT, UPDATE etc. See also the gettype() method.

4.5.13.1 Parameters

type

String whose value can be SELECT, UPDATE, INSERT or DELETE.

4.5.13.2 Syntax examples

```
$q = new xarQuery();
$q->settype('SELECT');
$q->addtable('xar_roles');
$q->eq('xar_name','Anonymous');
$q->qecho();

displays:

SELECT * FROM xar_roles WHERE xar_name = 'Anonymous'
```

4.5.14 uselimits()

The uselimits() method enables the LIMIT clause in a SELECT statement, if such a clause is defined. The SELECT statement will return the records defined by the setstartat() and setrowstodo() methods. By default limits are in effect if defined unless disabled by the nolimits() method.

4.5.14.1 Parameters

none

4.5.14.2 Syntax examples

```
$q = new xarQuery('SELECT','xar_security_levels');
$q->setstartat(3);
$q->setrowstodo(5);
$q->nolimits();
$q->run();
$total1 = count($q->output());
$q->uselimits();
$q->run();
$total2 = count($q->output());

returns:

$total1 = 10;    (all the records)
$total2 = 5;     (records 3 to 8)
```

4.6 Examples

Note: This chapter is out of date and will be rewritten!!

The following are equivalent:

```
        $dbconn = xarDBGetConn();
        $query = "SELECT * FROM $this->rolestable WHERE xar_name = '$this->name'";
        if(!$dbconn->Execute($query)) return;

        AND

        $q = new xarQuery('SELECT',$this->rolestable);
        $q->eq('xar_name',$this->name);
        if (!$q->run()) return;

        AND

        $q = new xarQuery();
        if (!$q->run("SELECT * FROM $this->rolestable WHERE xar_name =
'$this->name'")) return;
```

The following are equivalent:

```
        if (!$q->run($query)) return;
        return q->output();

        AND

        $result = $dbconn->Execute($query);
        if (!$result) return;
        $invoices = array();
        while(!$result->EOF) {
            list($this->id) = $result->fields;
            $invoices[] = $this->id;
            $result->MoveNext();
        }
        return $invoices;
```

Note that in the last code snippet some knowledge about the contents of $result is assumed.

Check if a role already exists:

```
        Old code:

        $this->dbconn = xarDBGetConn();
        // Confirm that this group or user does not already exist
        if ($this->type == 1) {
            $query = "SELECT COUNT(*) FROM $this->rolestable
                  WHERE xar_name = '$this->name'";
        } else {
            $query = "SELECT COUNT(*) FROM $this->rolestable
                  WHERE xar_uname = '$this->uname'";
        }

        $result = $this->dbconn->Execute($query);
        if (!$result) return;

        list($count) = $result->fields;

        if ($count == 1) {//error message}


        New code: Note only the WHERE clause changes depending on the role type

        // Confirm that this group or user does not already exist
        $q = new xarQuery('SELECT',$this->rolestable);
        if ($this->type == 1) {
            $q->eq('xar_name',$this->name);
        } else {
```

```
            $q->eq('xar_uname',$this->uname);
        }

        if (!$q->run()) return;

        if ($q->getrows() == 1) {//error message}
```

Adding a group or user to the roles table

Old code: Note the query code is repeated twice depending on whether the role to be added ia a user or a group.

```
$this->dbconn = xarDBGetConn();
$nextId = $this->dbconn->genID($this->rolestable);
$createdate = mktime();

if ($this->type == 1) {
    $query = "INSERT INTO $this->rolestable
                (xar_uid, xar_name, xar_type, xar_uname, xar_valcode, xar_date_reg)
              VALUES (?, ?, ?, ?, ?, ?)";
    $bindvars = array(
        $nextId,
        $this->name,
        $this->type,
        $this->uname,
        $this->val_code,
        $createdate,
    );
} else {
    $query = "INSERT INTO $this->rolestable
                (xar_uid, xar_name, xar_type, xar_uname, xar_email, xar_pass,
                xar_date_reg, xar_state, xar_valcode, xar_auth_module)
              VALUES (?,?,?,?,?,?,?,?,?,?)";
    $bindvars = array(
        $nextId,
        $this->name,
        $this->type,
        $this->uname,
        $this->email,
        md5($this->pass),
        $createdate,
        $this->state,
        $this->val_code,
        $this->auth_module,
    );
}
// Execute the query, bail if an exception was thrown
if (!$this->dbconn->Execute($query, $bindvars)) return;
```

New code: Note the query is assembled incrementally. The structured query handles the binding variables (? in the old code) internally.

```
$nextId = $this->dbconn->genID($this->rolestable);

$q = new xarQuery('INSERT',$this->rolestable);
$q->addfield(xar_uid,$nextId);
$q->addfield(xar_name,$this->name);
$q->addfield(xar_uname,$this->uname);
$q->addfield(xar_date_reg,mktime());
$q->addfield(xar_valcode,$this->val_code);
if ($this->type == 1) {
    $q->addfield(xar_type,1);
} else {
    $q->addfield(xar_type,0);
    $q->addfield(xar_email,$this->email);
    $q->addfield(xar_pass,md5($this->pass));
    $q->addfield(xar_state,$this->state);
    $q->addfield(xar_auth_module,$this->auth_module);
}
// Execute the query, bail if an exception was thrown
if (!$q->run()) return;
```

Author's Address

Marc Lutolf
Xaraya Development Group
EMail: marcinmilan@xaraya.com
URI: http://www.xaraya.com

A. Example appendix

Any section which is present after the references will become an appendix

Intellectual Property Statement

The DDF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the DDF's procedures with respect to rights in standards-track and standards-related documentation can be found in RFC-0.

The DDF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the DDF Board of Directors.

Acknowledgement

Funding for the RFC Editor function is provided by the DDF