MQdb::MappedQuery(3pm)

MQdb::MappedQuery(3pm) User Contribute

User Contributed Perl Documentation

NAME

MQdb::MappedQuery - DESCRIPTION of Object

SYNOPSIS

An Object_relational_mapping (ORM) design pattern based on mapping named_column results from any query into attributes of an object. As long as the column_names are parsable into attributes, any query is ok. This is an evolution of several ideas I have either used or created over the last 15 years of coding. This is a variation on the ActiveRecord design pattern but it trades more flexibility, power and control for slightly less automation. It still provides a development speed/ease advange over many ORM patterns.

DESCRIPTION

MappedQuery is an abstract superclass that is a variation on the ActiveRecord design pattern. Instead of actively mapping a table into an object, this will actively map the result of a query into an object. The query is standardized for a subclass of this object, and the columns returned by the query define the attributes of the object. This gives much more flexibility than the standard implementation of ActiveRecord. Since this design pattern is based around mapping a query (from potentially a multiple table join) to a single class object, this pattern is called MappedQuery.

In this particular implementation of this design pattern (mainly due to some limitations in perl) several aspects must be hand coded as part of the implementation of a subclass. Subclasses must handcode – all accessor methods – override the mapRow method – APIs for all explicit fetch methods

(by using the superclass fetch_single and fetch_multiple) – the store methods are coded by general DBI code (no framework assistance)

Individual MQdb::Database handle objects are assigned at an instance level for each object. This is different from some ActiveRecord implementations which place database handles into a global context or at the Class level. By placing it with each instance, this allows creation of instances of the same class pulled from two different databases, but with similar schemas. This is very useful when building certain types of data analysis systems.

The only restriction is that the database handle must be able run the queries that the object requires for it to work.

Future implementations could do more automatic code generation but this version already speeds development time by 2x-3x without imposing any limitations and retains all the flexibility of handcoding with DBI.

CONTACT

Jessica Severin < jessica DOT severin AT gmail DOT com>

LICENSE

- * Software License Agreement (BSD License)
- * MappedQueryDB [MQdb] toolkit
- * copyright (c) 2006-2009 Jessica Severin
- * All rights reserved.
- * Redistribution and use in source and binary forms, with or without
- modification, are permitted provided that the following conditions are met:
 - * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
 - * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
 - * Neither the name of Jessica Severin nor the
- * names of its contributors may be used to endorse or promote products* derived from this software without specific prior written permission.
- * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ''AS IS'' AND ANY
- * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
- * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
- * DISCLAIMED. IN NO EVENT SHALL COPYRIGHT HOLDERS BE LIABLE FOR ANY
- * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
- * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;



perl v5.10.0 2009-06-03 1

MQdb::MappedQuery(3pm)

User Contributed Perl Documentation

MQdb::MappedQuery(3pm)

- * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
- * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
- * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

APPENDIX

The rest of the documentation details each of the object methods. Internal methods are usually preceded with a _

mapRow

```
Description: This method must be overridden by subclasses to do the mapping of
             from the query response into attributes of the object. This is pa
             internal factory machinery. The instance of the class is created b
             called and the default init() method has already been called. The
             method is to initialize the rest of the state of the instance base
Arg (1)
           : $row_hash perl hash
Arg (2)
           : optional $dbc DBI connection (not generally used by most sublcasse
Returntype : $self
Exceptions : none
Caller
           : only called by internal factory methods
Example
                sub mapRow {
                    my $self = shift;
                    my $rowHash = shift;
                    $self->primary_id($rowHash->{'symbol_id'});
                    $self->type($rowHash->{'sym type'});
                    $self->symbol($rowHash->{'sym_value'});
                    return $self;
                }
```

store

Description: This method is just an empty template as part of the API definition. How it is defined, and how parameters are handled are completely usubclass. Each subclass should override and implement.

Returntype : \$self Exceptions : none

Caller : general loader scripts

}

fetch_single

```
Description: General purpose template method for fetching a single instance
             of this class(subclass) using the mapRow method to convert
             a row of data into an object.
           : $database (MQdb::Database)
Arg (1)
Arg (2) : $sql (string of SQL statement with place holders)
Arg (3...): optional parameters to map to the placehodlers within the SQL
Returntype : instance of this Class (subclass)
Exceptions : none
        : subclasses (not public methods)
Caller
Example
                sub fetch_by_id {
                   my $class = shift;
                    my $db = shift;
                    my $id = shift;
                    my $sql = "SELECT * FROM symbol WHERE symbol_id=?";
                    return $class->fetch_single($db, $sql, $id);
```

fetch_multiple



perl v5.10.0 2009-06-03 2

```
MQdb::MappedQuery(3pm)
```

```
User Contributed Perl Documentation
                                                      MQdb::MappedQuery(3pm)
     Description: General purpose template method for fetching an array of instance
                  of this class(subclass) using the mapRow method to convert
                  a row of data into an object.
                : $database (MQdb::Database)
     Arg (1)
              : $sql (string of SQL statement with place holders)
     Arg (2)
     Arg (3...): optional parameters to map to the placehodlers within the SQL
     Returntype: array of all instances of this Class (subclass) which match the qu
     Exceptions : none
               : subclasses (not public methods)
     Example
                      sub fetch_all_by_value {
                         my $class = shift;
                         my $db = shift;
                         my $name = shift;
                         my $sql = "SELECT * FROM symbol WHERE sym_value=?";
                         return $class->fetch_multiple($db, $sql, $name);
                      }
stream_multiple
     Description: General purpose template method for fetching multiple instance
                  of this class(subclass) using the mapRow method to convert
                  a row of data into an object. Instead of instantiating all
                  instance at once and returning as array, this method returns
                  a DBStream instance which then creates each instance from an
                  open handle on each $stream->next_in_stream() call.
     Arg (1)
                : $database (MQdb::Database)
               : $sql (string of SQL statement with place holders)
     Arg (2)
     Arg (3...): optional parameters to map to the placehodlers within the SQL
     Returntype : DBStream object
     Exceptions : none
     Caller : subclasses use this internally when creating new API stream_by....
     Example
                     sub stream_by_value {
                         my $class = shift;
                         my $db = shift;
                         my $name = shift;
                         my $sql = "SELECT * FROM symbol WHERE sym_value=?";
                         return $class->stream_multiple($db, $sql, $name);
                      }
fetch_col_value
     Description: General purpose function to allow fetching of a single column from
     Arg (1) : $sql (string of SQL statement with place holders)
     Arg (2...): optional parameters to map to the placehodlers within the SQL
     Example
               : $value = $self->fetch_col_value($db,
                                                   "select symbol_id from symbol wher
                                                   $type,$value);
     Returntype : scalar value
     Exceptions : none
     Caller
               : within subclasses to easy development
fetch_col_array
     Description: General purpose function to allow fetching of a single column from
     Arg (1) : $sql (string of SQL statement with place holders)
```

```
Arg (2...): optional parameters to map to the placehodlers within the SQL
Example : $array_ref = $self->fetch_col_array($db, "select some_column from")
Returntype : array reference of scalar values
```

Exceptions : none

: within subclasses to easy development



perl v5.10.0 2009-06-03 3 MQdb::MappedQuery(3pm) User Contributed Perl Documentation MQdb::MappedQuery(3pm)

next_sequence_id

Description: Convenience method for working with SEQUENCES in ORACLE databases.

Arg (1) : \$sequenceName

Returntype : scalar of the nextval in the sequence

Exceptions : none



perl v5.10.0 2009-06-03 4