Zend_Db_Cursor - Bruno Piraja Moyle

<ac:macro ac:name="info"><ac:parameter ac:name="title">New Proposal Template</ac:parameter><ac:rich-text-body>
This page has been created from a template that uses "zones." To proceed:

Edit the page
Replace sample content within each zone-data tag with your own content
Remove this notice
Save the page
When you are ready for community review, move this page to the <ac:link><ri:page ri:content-title="Ready for Review" /></ac:link> section on the edit page.

<ac:macro ac:name="note"><ac:parameter ac:name="title">No placeholders allowed!</ac:parameter><ac:rich-text-body>
Please do not create placeholders. Wait until you have sufficient content to replace all sample data in the proposal template before creating your proposal document. </ac:rich-text-body></ac:macro></ac:rich-text-body></ac:macro>

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Db_Cursor Component Proposal

Proposed Component Name	Zend_Db_Cursor
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Db_Cursor
Proposers	Bruno Piraja Moyle
Zend Liaison	TBD
Revision	1.0 - 24 june 2009: Initial Draft. (wiki revision: 17)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

Zend_Db_Cursor goal is to allow using database ref cursors in a row set flavor. Most of Oracle stored procedures makes use of ref cursors and usually returns it as an 'in out' parameter. oci8 extension allow the creation and access of these cursors.

2. References

- oci8 at php.net

3. Component Requirements, Constraints, and Acceptance Criteria

These components depends that PHP specific database extension provides interfaces to create these kind of cursors, as oci8 does.

4. Dependencies on Other Framework Components

- Zend_Exception
- Zend_Db_Adapter
- Zend_Db_Statement

5. Theory of Operation

The component should be used as a wrapper for cursors, by instantiating an object using the new operator:

```
// Create an Oracle statement
$stmt = new Zend_Db_Statement_Oracle(
    $db, 'begin open_all_emp(:cursor); end;'
);

// Create a cursor
$cursor = new Zend_Db_Cursor_Oracle($db);

// Bind the cursor as a parameter. This SHOULD push a new cursor in the
// $_bindCursor stack from Zend_Db_Statement.
$stmt->bindCursor('cursor', $cursor);

// Execute the statement. This SHOULD check the count() of the $_bindCursor
// stack from Zend_Db_Statement and execute each of them.
$stmt->execute();

// Both statement and cursor are executed. As Zend_Db_Cursor implements Iterator and Countable
// interface, is possible to access it in a row set flavor
$this->view->cursor = $cursor;

// Debugging
//Zend_Debug::dump($cursor);

// Render 'index' view script
```

In the 'index' view script

```
<h1>Employees</h1>
<?php if (count($this->cursor)) : ?>
    <?php foreach ($this->cursor as $row) : ?>
        <p><?php echo $row->empno ?> - <?php echo $row['ename'] ?></p>
    <?php endforeach ?>
<?php else : ?>
    <p>Empty cursor...</p>
<?php endif ?>
<address>Powered by Zend Framework <?php echo Zend_Version::VERSION ?></address>
```

To test the component, you can use the following stored procedure - in Oracle(r) database:

```
create or replace procedure open_all_emp(p_emp_refcur in out sys_refcursor)
is
begin
    open p_emp_refcur for select empno, ename from emp;
end;
```

This stored procedure receives an 'in out' ref cursor as a parameter and open it. By opening a ref cursor, it is populated and is ready to fetch.

6. Milestones / Tasks

- Milestone 1: [Done] Component development;
- Milestone 2: Tests;
- Milestone 3: Documentation;
- Milestone 4: Approval.

7. Class Index

- Zend_Db_Cursor_CursorAbstract
- Zend_Db_Cursor_Oracle
- Zend_Db_Cursor_Row
- Zend_Db_Cursor_Exception
- Zend_Db_Statement (this class needs to be updated in order to be aware of ref cursors binding)

8. Use Cases

UC-01

9. Class Skeletons

```
<?php
/**
 * Zend Framework
 *
 * LICENSE
 *
 * This source file is subject to the new BSD license that is bundled
 * with this package in the file LICENSE.txt.
 * It is also available through the world-wide-web at this URL:
 * http://framework.zend.com/license/new-bsd
 * If you did not receive a copy of the license and are unable to
 * obtain it through the world-wide-web, please send an email
 * to license@zend.com so we can send you a copy immediately.
 *
 * @category   Zend
 * @package    Zend_Db
 * @subpackage Statement
```

```
 * @copyright  Copyright (c) 2005-2009 Zend Technologies USA Inc. (http://www.zend.com)
 * @license    http://framework.zend.com/license/new-bsd     New BSD License
 * @version    $Id: Statement.php 18951 2009-11-12 16:26:19Z alexander $
 */

/**
 * @see Zend_Db
 */
require_once 'Zend/Db.php';

/**
 * @see Zend_Db_Statement_Interface
 */
require_once 'Zend/Db/Statement/Interface.php';

/**
 * Abstract class to emulate a PDOStatement for native database adapters.
 *
 * @category   Zend
 * @package    Zend_Db
 * @subpackage Statement
 * @copyright  Copyright (c) 2005-2009 Zend Technologies USA Inc. (http://www.zend.com)
 * @license    http://framework.zend.com/license/new-bsd     New BSD License
 */
abstract class Zend_Db_Statement implements Zend_Db_Statement_Interface
{

    /**
     * @var resource|object The driver level statement object/resource
     */
    protected $_stmt = null;

    /**
     * @var Zend_Db_Adapter_Abstract
     */
    protected $_adapter = null;

    /**
     * The current fetch mode.
     *
     * @var integer
     */
    protected $_fetchMode = Zend_Db::FETCH_ASSOC;

    /**
     * Attributes.
     *
     * @var array
     */
    protected $_attribute = array();

    /**
     * Column result bindings.
     *
     * @var array
     */
    protected $_bindColumn = array();

    /**
     * Query parameter bindings; covers bindParam() and bindValue().
     *
     * @var array
     */
    protected $_bindParam = array();

    /**
     * Cursors binding.
     *
     * @var array
     */
```

```
    protected $_bindCursor = array();


    /**
     * SQL string split into an array at placeholders.
     *
     * @var array
     */
    protected $_sqlSplit = array();

    /**
     * Parameter placeholders in the SQL string by position in the split array.
     *
     * @var array
     */
    protected $_sqlParam = array();

    /**
     * @var Zend_Db_Profiler_Query
     */
    protected $_queryId = null;

    /**
     * Constructor for a statement.
     *
     * @param Zend_Db_Adapter_Abstract $adapter
     * @param mixed $sql Either a string or Zend_Db_Select.
     */
    public function __construct($adapter, $sql)
    {
        $this->_adapter = $adapter;
        if ($sql instanceof Zend_Db_Select) {
            $sql = $sql->assemble();
        }
        $this->_parseParameters($sql);
        $this->_prepare($sql);

        $this->_queryId = $this->_adapter->getProfiler()->queryStart($sql);
    }

    /**
     * Internal method called by abstract statment constructor to setup
     * the driver level statement
     *
     * @return void
     */
    protected function _prepare($sql)
    {
        return;
    }

    /**
     * @param string $sql
     * @return void
     */
    protected function _parseParameters($sql)
    {
        $sql = $this->_stripQuoted($sql);

        // split into text and params
        $this->_sqlSplit = preg_split('/(\?|\:[a-zA-Z0-9_]+)/',
            $sql, -1, PREG_SPLIT_DELIM_CAPTURE|PREG_SPLIT_NO_EMPTY);

        // map params
        $this->_sqlParam = array();
        foreach ($this->_sqlSplit as $key => $val) {
            if ($val == '?') {
                if ($this->_adapter->supportsParameters('positional') === false) {
                    /**
                     * @see Zend_Db_Statement_Exception
```

```
                 */
                require_once 'Zend/Db/Statement/Exception.php';
                throw new Zend_Db_Statement_Exception("Invalid bind-variable position '$val'");
            }
        } else if ($val[0] == ':') {
            if ($this->_adapter->supportsParameters('named') === false) {
                /**
                 * @see Zend_Db_Statement_Exception
                 */
                require_once 'Zend/Db/Statement/Exception.php';
                throw new Zend_Db_Statement_Exception("Invalid bind-variable name '$val'");
            }
        }
        $this->_sqlParam[] = $val;
    }

    // set up for binding
    $this->_bindParam = array();
}

/**
 * Remove parts of a SQL string that contain quoted strings
 * of values or identifiers.
 *
 * @param string $sql
 * @return string
 */
protected function _stripQuoted($sql)
{
    // get the character for delimited id quotes,
    // this is usually " but in MySQL is `
    $d = $this->_adapter->quoteIdentifier('a');
    $d = $d[0];

    // get the value used as an escaped delimited id quote,
    // e.g. \" or "" or \`
    $de = $this->_adapter->quoteIdentifier($d);
    $de = substr($de, 1, 2);
    $de = str_replace('\\', '\\\\', $de);

    // get the character for value quoting
    // this should be '
    $q = $this->_adapter->quote('a');
    $q = $q[0];

    // get the value used as an escaped quote,
    // e.g. \' or ''
    $qe = $this->_adapter->quote($q);
    $qe = substr($qe, 1, 2);
    $qe = str_replace('\\', '\\\\', $qe);

    // get a version of the SQL statement with all quoted
    // values and delimited identifiers stripped out
    // remove "foo\"bar"
    $sql = preg_replace("/$q($qe|\\\\{2}|[^$q])*$q/", '', $sql);
    // remove 'foo\'bar'
    if (!empty($q)) {
        $sql = preg_replace("/$q($qe|[^$q])*$q/", '', $sql);
    }

    return $sql;
}

/**
 * Bind a column of the statement result set to a PHP variable.
 *
 * @param string $column Name the column in the result set, either by
 *                       position or by name.
 * @param mixed  $param  Reference to the PHP variable containing the value.
 * @param mixed  $type   OPTIONAL
```

```
     * @return bool
     */
    public function bindColumn($column, &$param, $type = null)
    {
        $this->_bindColumn[$column] =& $param;
        return true;
    }

    /**
     * Binds a parameter to the specified variable name.
     *
     * @param mixed $parameter Name the parameter, either integer or string.
     * @param mixed $variable  Reference to PHP variable containing the value.
     * @param mixed $type      OPTIONAL Datatype of SQL parameter.
     * @param mixed $length    OPTIONAL Length of SQL parameter.
     * @param mixed $options   OPTIONAL Other options.
     * @return bool
     */
    public function bindParam($parameter, &$variable, $type = null, $length = null, $options = null)
    {
        if (!is_int($parameter) && !is_string($parameter)) {
            /**
             * @see Zend_Db_Statement_Exception
             */
            require_once 'Zend/Db/Statement/Exception.php';
            throw new Zend_Db_Statement_Exception('Invalid bind-variable position');
        }

        $position = null;
        if (($intval = (int) $parameter) > 0 && $this->_adapter->supportsParameters('positional')) {
            if ($intval >= 1 || $intval <= count($this->_sqlParam)) {
                $position = $intval;
            }
        } else if ($this->_adapter->supportsParameters('named')) {
            if ($parameter[0] != ':') {
                $parameter = ':' . $parameter;
            }
            if (in_array($parameter, $this->_sqlParam) !== false) {
                $position = $parameter;
            }
        }

        if ($position === null) {
            /**
             * @see Zend_Db_Statement_Exception
             */
            require_once 'Zend/Db/Statement/Exception.php';
            throw new Zend_Db_Statement_Exception("Invalid bind-variable position '$parameter'");
        }

        // Finally we are assured that $position is valid
        $this->_bindParam[$position] =& $variable;
        return $this->_bindParam($position, $variable, $type, $length, $options);
    }

    /**
     * Binds a value to a parameter.
     *
     * @param mixed $parameter Name the parameter, either integer or string.
     * @param mixed $value     Scalar value to bind to the parameter.
     * @param mixed $type      OPTIONAL Datatype of the parameter.
     * @return bool
     */
    public function bindValue($parameter, $value, $type = null)
    {
        return $this->bindParam($parameter, $value, $type);
    }

    /**
     * Binds a ref cursor as a parameter.
```

```
     *
     * @param  string $parameter
     * @param  Zend_Db_Cursor_CursorAbstract $cursor
     * @return void
     */
    public function bindCursor($parameter, Zend_Db_Cursor_CursorAbstract &$cursor)
    {
        // Push the cursor in the cursors binding property
        array_push($this->_bindCursor, $cursor);

        // Bind as a parameter
        $this->_bindParam($parameter, $cursor->getRawCursor(), OCI_B_CURSOR, -1);
    }

    /**
     * Executes a prepared statement.
     *
     * @param array $params OPTIONAL Values to bind to parameter placeholders.
     * @return bool
     */
    public function execute(array $params = null)
    {
        /*
         * Simple case - no query profiler to manage.
         */
        if ($this->_queryId === null) {
            // Get the execute return value
            $retval = $this->_execute($params);

            // Execute any binded ref cursor
            if (count($this->_bindCursor)) {
                foreach ($this->_bindCursor as $cursor) {
                    $cursor->execute();
                }
            }

            return $retval;
        }

        /*
         * Do the same thing, but with query profiler
         * management before and after the execute.
         */
        $prof = $this->_adapter->getProfiler();
        $qp = $prof->getQueryProfile($this->_queryId);
        if ($qp->hasEnded()) {
            $this->_queryId = $prof->queryClone($qp);
            $qp = $prof->getQueryProfile($this->_queryId);
        }
        if ($params !== null) {
            $qp->bindParams($params);
        } else {
            $qp->bindParams($this->_bindParam);
        }
        $qp->start($this->_queryId);

        $retval = $this->_execute($params);

        $prof->queryEnd($this->_queryId);

        return $retval;
    }

    /**
     * Returns an array containing all of the result set rows.
     *
     * @param int $style OPTIONAL Fetch mode.
     * @param int $col   OPTIONAL Column number, if fetch mode is by column.
     * @return array Collection of rows, each in a format by the fetch mode.
     */
```

```
    public function fetchAll($style = null, $col = null)
    {
        $data = array();
        if ($style === Zend_Db::FETCH_COLUMN && $col === null) {
            $col = 0;
        }
        if ($col === null) {
            while ($row = $this->fetch($style)) {
                $data[] = $row;
            }
        } else {
            while (false !== ($val = $this->fetchColumn($col))) {
                $data[] = $val;
            }
        }
        return $data;
    }

    /**
     * Returns a single column from the next row of a result set.
     *
     * @param int $col OPTIONAL Position of the column to fetch.
     * @return string One value from the next row of result set, or false.
     */
    public function fetchColumn($col = 0)
    {
        $data = array();
        $col = (int) $col;
        $row = $this->fetch(Zend_Db::FETCH_NUM);
        if (!is_array($row)) {
            return false;
        }
        return $row[$col];
    }

    /**
     * Fetches the next row and returns it as an object.
     *
     * @param string $class  OPTIONAL Name of the class to create.
     * @param array  $config OPTIONAL Constructor arguments for the class.
     * @return mixed One object instance of the specified class, or false.
     */
    public function fetchObject($class = 'stdClass', array $config = array())
    {
        $obj = new $class($config);
        $row = $this->fetch(Zend_Db::FETCH_ASSOC);
        if (!is_array($row)) {
            return false;
        }
        foreach ($row as $key => $val) {
            $obj->$key = $val;
        }
        return $obj;
    }

    /**
     * Retrieve a statement attribute.
     *
     * @param string $key Attribute name.
     * @return mixed      Attribute value.
     */
    public function getAttribute($key)
    {
        if (array_key_exists($key, $this->_attribute)) {
            return $this->_attribute[$key];
        }
    }

    /**
     * Set a statement attribute.
```

```
     *
     * @param string $key Attribute name.
     * @param mixed  $val Attribute value.
     * @return bool
     */
    public function setAttribute($key, $val)
    {
        $this->_attribute[$key] = $val;
    }

    /**
     * Set the default fetch mode for this statement.
     *
     * @param int   $mode The fetch mode.
     * @return bool
     * @throws Zend_Db_Statement_Exception
     */
    public function setFetchMode($mode)
    {
        switch ($mode) {
            case Zend_Db::FETCH_NUM:
            case Zend_Db::FETCH_ASSOC:
            case Zend_Db::FETCH_BOTH:
            case Zend_Db::FETCH_OBJ:
                $this->_fetchMode = $mode;
                break;
            case Zend_Db::FETCH_BOUND:
            default:
                $this->closeCursor();
                /**
                 * @see Zend_Db_Statement_Exception
                 */
                require_once 'Zend/Db/Statement/Exception.php';
                throw new Zend_Db_Statement_Exception('invalid fetch mode');
                break;
        }
    }

    /**
     * Helper function to map retrieved row
     * to bound column variables
     *
     * @param array $row
     * @return bool True
     */
    public function _fetchBound($row)
    {
        foreach ($row as $key => $value) {
            // bindColumn() takes 1-based integer positions
            // but fetch() returns 0-based integer indexes
            if (is_int($key)) {
                $key++;
            }
            // set results only to variables that were bound previously
            if (isset($this->_bindColumn[$key])) {
                $this->_bindColumn[$key] = $value;
            }
        }
        return true;
    }

    /**
     * Gets the Zend_Db_Adapter_Abstract for this
     * particular Zend_Db_Statement object.
     *
     * @return Zend_Db_Adapter_Abstract
     */
    public function getAdapter()
    {
        return $this->_adapter;
```

```
    }

    /**
     * Gets the resource or object setup by the
     * _parse
     * @return unknown_type
     */
    public function getDriverStatement()
    {
        return $this->_stmt;
    }
}
```

```
<?php
/**
 * Zend Framework
 *
 * LICENSE
 *
 * This source file is subject to the new BSD license that is bundled
 * with this package in the file LICENSE.txt.
 * It is also available through the world-wide-web at this URL:
 * http://framework.zend.com/license/new-bsd
 * If you did not receive a copy of the license and are unable to
 * obtain it through the world-wide-web, please send an email
 * to license@zend.com so we can send you a copy immediately.
 *
 * @category   Zend
 * @package    Zend_Db
 * @subpackage Cursor
 * @copyright  Copyright (c) 2005-2010 Zend Technologies USA Inc. (http://www.zend.com)
 * @license    http://framework.zend.com/license/new-bsd     New BSD License
 * @version    $Id: Oracle.php 18951 2010-01-12 16:26:19Z brunitto $
 */

/**
 * Oracle database cursors.
 *
 * @package    Zend_Db
 * @subpackage Cursor
 * @author     brunitto
 */
class Zend_Db_Cursor_Oracle extends Zend_Db_Cursor
{
    /**
     * Constructor.
     *
     * Receive a database adapter and try to allocate a new OCI cursor.
     *
     * @param  Zend_Db_Adapter_Oracle $db
     * @throws Zend_Db_Cursor_Exception
     */
    public function __construct(Zend_Db_Adapter_Oracle $db)
    {
        // Get the OCI connection resource and a cursor
        $conn = $db->getConnection();
        $cursor = oci_new_cursor($conn);

        // Check if the cursor is a valid resource
        if (!is_resource($cursor)) {
            require_once 'Zend/Db/Cursor/Exception.php';
            throw new Zend_Db_Cursor_Exception('Can\'t allocate a cursor.');
        }

        // Setup $_cursor property
        $this->_cursor = $cursor;
    }
```

```
    public function execute()
    {
        if (!oci_execute($this->_cursor)) {
            require_once 'Zend/Db/Cursor/Exception.php';
            throw new Zend_Db_Cursor_Exception('Couldn\'t execute the cursor.');
        }

        // Cursor is ready to go
        while ($data = oci_fetch_assoc($this->_cursor)) {
            require_once 'Zend/Db/Cursor/Row.php';
            $row = new Zend_Db_Cursor_Row();

            foreach ($data as $key => $value) {
                $row->$key = $value;
            }

            $this->pushRow($row);
        }
    }

    public function free()
    {
        if (!oci_free_statement($this->_cursor)) {
            require_once 'Zend/Db/Cursor/Exception.php';
            throw new Zend_Db_Cursor_Exception('Couldn\'t free the cursor.');
        }

        // Reset the cursor
        $this->_cursor = null;
    }
}




<?php
/**
 * Zend Framework
 *
 * LICENSE
 *
 * This source file is subject to the new BSD license that is bundled
 * with this package in the file LICENSE.txt.
 * It is also available through the world-wide-web at this URL:
 * http://framework.zend.com/license/new-bsd
 * If you did not receive a copy of the license and are unable to
 * obtain it through the world-wide-web, please send an email
 * to license@zend.com so we can send you a copy immediately.
 *
 * @category   Zend
 * @package    Zend_Db
 * @subpackage Cursor
 * @copyright  Copyright (c) 2005-2010 Zend Technologies USA Inc. (http://www.zend.com)
 * @license    http://framework.zend.com/license/new-bsd     New BSD License
 * @version    $Id: Cursor.php 18951 2010-01-12 16:26:19Z brunitto $
 */

/**
 * Concrete class for general database cursors.
 *
 * @package    Zend_Db
 * @subpackage Cursor
 * @author     brunitto
 */
class Zend_Db_Cursor extends Zend_Db_Cursor_CursorAbstract
{
}
```

```
<?php
/**
 * Zend Framework
 *
 * LICENSE
 *
 * This source file is subject to the new BSD license that is bundled
 * with this package in the file LICENSE.txt.
 * It is also available through the world-wide-web at this URL:
 * http://framework.zend.com/license/new-bsd
 * If you did not receive a copy of the license and are unable to
 * obtain it through the world-wide-web, please send an email
 * to license@zend.com so we can send you a copy immediately.
 *
 * @category   Zend
 * @package    Zend_Db
 * @subpackage Cursor
 * @copyright  Copyright (c) 2005-2010 Zend Technologies USA Inc. (http://www.zend.com)
 * @license    http://framework.zend.com/license/new-bsd     New BSD License
 * @version    $Id: Abstract.php 18951 2010-01-12 16:26:19Z brunitto $
 */

/**
 * Abstract database cursors.
 *
 * @package    Zend_Db
 * @subpackage Cursor
 * @author     brunitto
 */
abstract class Zend_Db_Cursor_CursorAbstract implements Countable, Iterator
{
    /**
     * Database specific cursor.
     *
     * @var resource
     */
    protected $_cursor = null;

    /**
     * Cursor rows.
     *
     * @var array
     */
    protected $_rows   = array();

    /**
     * Rows offset. Useful for Iterator interface methods.
     *
     * @var integer
     */
    protected $_offset = 0;

    /**
     * Return the rows count. Required by the Iterator implementation.
     *
     * @return integer
     */
    public function count()
    {
        return count($this->_rows);
    }

    /**
     * Return the current row. Required by Iterator implementation.
     *
     * @return mixed
     */
    public function current()
```

```
    {
        return $this->_rows[$this->_offset];
    }

    /**
     * Return the current row key. Required by Iterator implementation.
     *
     * @return integer
     */
    public function key()
    {
        return $this->_offset;
    }

    /**
     * Increment the rows offset.
     *
     * @return void
     */
    public function next()
    {
        ++$this->_offset;
    }

    /**
     * Rewind the rows offset.
     *
     * @return void
     */
    public function rewind()
    {
        $this->_offset = 0;
    }

    /**
     * Return if the current row is valid.
     *
     * @return boolean
     */
    public function valid()
    {
        return isset($this->_rows[$this->_offset]);
    }

    /**
     * Pop a row from cursor.
     *
     * @return mixed
     */
    public function popRow()
    {
        return array_pop($this->_rows);
    }

    /**
     * Push a row to cursor.
     *
     * @param  Zend_Db_Cursor_Row $row
     * @return boolean
     */
    public function pushRow(Zend_Db_Cursor_Row $row)
    {
        return array_push($this->_rows, $row);
    }

    /**
     * Free a cursor.
     *
     * @return void
     */
```

```
    public function free()
    {
        // Method body defined in specific classes
    }

    /**
     * Execute the cursor.
     *
     * @return void
     */
    public function execute()
    {
        // Method body defined in specific classes
    }

    /**
     * Get the database cursor.
     *
     * @return resource
     */
    public function getRawCursor()
    {
        return $this->_cursor;
    }
}
```

```
<?php
/**
 * Zend Framework
 *
 * LICENSE
 *
 * This source file is subject to the new BSD license that is bundled
 * with this package in the file LICENSE.txt.
 * It is also available through the world-wide-web at this URL:
 * http://framework.zend.com/license/new-bsd
 * If you did not receive a copy of the license and are unable to
 * obtain it through the world-wide-web, please send an email
 * to license@zend.com so we can send you a copy immediately.
 *
 * @category   Zend
 * @package    Zend_Db
 * @subpackage Cursor
 * @copyright  Copyright (c) 2005-2010 Zend Technologies USA Inc. (http://www.zend.com)
 * @license    http://framework.zend.com/license/new-bsd     New BSD License
 * @version    $Id: Exception.php 18951 2010-01-12 16:26:19Z brunitto $
 */

/**
 * Database cursors exception class.
 *
 * @package    Zend_Db
 * @subpackage Cursor
 * @author     brunitto
 */
class Zend_Db_Cursor_Exception extends Zend_Exception
{
}
```

```
<?php
/**
 * Zend Framework
 *
 * LICENSE
```

```
 *
 * This source file is subject to the new BSD license that is bundled
 * with this package in the file LICENSE.txt.
 * It is also available through the world-wide-web at this URL:
 * http://framework.zend.com/license/new-bsd
 * If you did not receive a copy of the license and are unable to
 * obtain it through the world-wide-web, please send an email
 * to license@zend.com so we can send you a copy immediately.
 *
 * @category   Zend
 * @package    Zend_Db
 * @subpackage Cursor
 * @copyright  Copyright (c) 2005-2010 Zend Technologies USA Inc. (http://www.zend.com)
 * @license    http://framework.zend.com/license/new-bsd     New BSD License
 * @version    $Id: Row.php 18951 2010-01-12 16:26:19Z brunitto $
 */

/**
 * Database cursors row class.
 *
 * @package    Zend_Db_Cursor
 * @subpackage Cursor
 * @author     brunitto
 */
class Zend_Db_Cursor_Row implements ArrayAccess
{
    /**
     * Data array as column => value.
     *
     * @var array
     */
    protected $_data = array();

    public function __get($column)
    {
        $column = strtolower($column);
        if (!array_key_exists($this->_data, $column)) {
            require_once 'Zend/Db/Cursor/Exception.php';
            throw new Zend_Db_Cursor_Exception('Invalid column.');
        }

        return $this->_data[$column];
    }

    public function __set($column, $value)
    {
        $column               = strtolower($column);
        $this->_data[$column] = $value;
    }

    public function offsetExists($offset)
    {
        return isset($this->_data[$offset]);
    }

    public function offsetGet($offset)
    {
        return $this->_data[$offset];
    }

    public function offsetSet($offset, $value)
    {
        $this->_data[$offset] = $value;
    }

    public function offsetUnset($offset)
    {
```

```
            unset($this->_data[$offset]);
        }
    }
```

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>