```php
<?php namespace System\Database;

/**
 * Nano
 *
 * Just another php framework
 *
 * @package		nano
 * @link		http://madebykieron.co.uk
 * @copyright	http://unlicense.org/
 */

use PDO;
use Closure;
use System\Database as DB;

class Query extends Builder {

	/**
	 * The current database table
	 *
	 * @var string
	 */
	public $table;

	/**
	 * Database connector object
	 *
	 * @var object
	 */
	public $connection;
```

```
/**
 * The class name of the object to create when fetching from the database
 *
 * @var string
 */
public $fetch_class = 'StdClass';

/**
 * Array of columns to build the select
 *
 * @var array
 */
public $columns = array();

/**
 * Array of table joins to build
 *
 * @var array
 */
public $join = array();

/**
 * Array of where clauses to build
 *
 * @var array
 */
public $where = array();

/**
 * Columns to sort by
 *
 * @var array
```

```
 */
public $sortby = array();

/**
 * Columns to group by
 *
 * @var array
 */
public $groupby = array();

/**
 * Number of rows to limit
 *
 * @var int
 */
public $limit;

/**
 * Number of rows to offset
 *
 * @var int
 */
public $offset;

/**
 * Array values to bind to the query
 *
 * @var array
 */
public $bind = array();

/**
```

```
     * Create a new database query instance for chaining
     *
     * @param string
     * @param object Connector
     * @return object Query
     */
    public static function table($table, $connection = null) {
            if(is_null($connection)) $connection = DB::connection();

            return new static($table, $connection);
    }

    /**
     * Create a new database query instance
     *
     * @param string
     * @param object Connector
     */
    public function __construct($table, $connection = null) {
            if(is_null($connection)) $connection = DB::connection();

            $this->table = $table;
            $this->connection = $connection;
    }

    /**
     * Set the class name for fetch queries, return self for chaining
     *
     * @param string
     * @return object
     */
    public function apply($class) {
```

```
            $this->fetch_class = $class;

            return $this;
        }

        /**
         * Run a count function on database query
         *
         * @return string
         */
        public function count() {
            list($result, $statement) = $this->connection->ask($this->build_select_count(), $this->bind);

            return $statement->fetchColumn();
        }

        /**
         * Fetch a single column from the query
         *
         * @param array
         * @param int
         * @return string
         */
        public function column($columns = array(), $column_number = 0) {
            list($result, $statement) = $this->connection->ask($this->build_select($columns), $this->bind);

            return $statement->fetchColumn($column_number);
        }

        /**
         * Fetch a single row from the query
```

```
         *
         * @param array
         * @return object
         */
        public function fetch($columns = null) {
                list($result, $statement) = $this->connection->ask($this->build_select($columns), $this->bind);

                $statement->setFetchMode(PDO::FETCH_CLASS|PDO::FETCH_PROPS_LATE, $this->fetch_class);

                return $statement->fetch();
        }

        /**
         * Fetch a result set from the query
         *
         * @param array
         * @return object
         */
        public function get($columns = null) {
                list($result, $statement) = $this->connection->ask($this->build_select($columns), $this->bind);

                $statement->setFetchMode(PDO::FETCH_CLASS|PDO::FETCH_PROPS_LATE, $this->fetch_class);

                return $statement->fetchAll();
        }

        /**
         * Insert a row into the database
         *
```

```
     * @param array
     * @return object
     */
    public function insert($row) {
        list($result, $statement) = $this->connection->ask($this->build_insert($row), $this->bind);

        return $statement->rowCount();
    }

    /**
     * Insert a row into the database and return the inserted ID
     *
     * @param array
     * @return int
     */
    public function insert_get_id($row) {
        list($result, $statement) = $this->connection->ask($this->build_insert($row), $this->bind);

        return $this->connection->instance()->lastInsertId();
    }

    /**
     * Update row in the database
     *
     * @param array
     * @return int
     */
    public function update($row) {
        list($result, $statement) = $this->connection->ask($this->build_update($row), $this->bind);

        return $statement->rowCount();
    }
```

```
	/**
	 * Delete a row in the database
	 *
	 * @return int
	 */
	public function delete() {
		list($result, $statement) = $this->connection->ask($this->build_delete(), $this->bind);

		return $statement->rowCount();
	}

	/**
	 * Add a where clause to the query
	 *
	 * @param string
	 * @param string
	 * @param string
	 * @return object
	 */
	public function where($column, $operator, $value) {
		$this->where[] = (count($this->where) ? 'AND ' : 'WHERE ') . $this->wrap($column) . ' ' .
$operator . ' ?';
		$this->bind[] = $value;

		return $this;
	}

	/**
	 * Add a where clause to the query starting with OR
	 *
	 * @param string
```

```
         * @param string
         * @param string
         * @return object
         */
        public function or_where($column, $operator, $value) {
                $this->where[] = (count($this->where) ? 'OR ' : 'WHERE ') . $this->wrap($column) . ' ' .
$operator . ' ?';
                $this->bind[] = $value;

                return $this;
        }

        /**
         * Add a where clause to the query starting with IN
         *
         * @param string
         * @param array
         * @return object
         */
        public function where_in($column, $values) {
                $this->where[] = (count($this->where) ? 'OR ' : 'WHERE ') .
                        $this->wrap($column) . ' IN (' . $this->placeholders(count($values)) . ')';

                $this->bind = array_merge($this->bind, $values);

                return $this;
        }

        /**
         * Add a table join to the query
         *
         * @param string|function
```

```
	 * @param string
	 * @param string
	 * @param string
	 * @param string
	 * @return object
	 */
	public function join($table, $left, $operator, $right, $type = 'INNER') {
		if($table instanceof \Closure) {
			list($query, $alias) = $table();

			$this->bind = array_merge($this->bind, $query->bind);

			$table = '(' . $query->build_select() . ') AS ' . $this->wrap($alias);
		}
		else $table = $this->wrap($table);

		$this->join[] = $type . ' JOIN ' . $table . ' ON (' . $this->wrap($left) . ' ' . $operator . ' ' . $this->wrap($right) . ')';

		return $this;
	}

	/**
	 * Add a left table join to the query
	 *
	 * @param string
	 * @param string
	 * @param string
	 * @param string
	 * @return object
	 */
	public function left_join($table, $left, $operator, $right) {
```

```
        return $this->join($table, $left, $operator, $right, 'LEFT');
}

/**
 * Add a sort by column to the query
 *
 * @param string
 * @param string
 * @return object
 */
public function sort($column, $mode = 'ASC') {
        $this->sortby[] = $this->wrap($column) . ' ' . strtoupper($mode);

        return $this;
}

/**
 * Add a group by column to the query
 *
 * @param string
 * @return object
 */
public function group($column) {
        $this->groupby[] = $this->wrap($column);

        return $this;
}

/**
 * Set a row limit on the query
 *
 * @param int
```

```
     * @return object
     */
    public function take($num) {
        $this->limit = $num;

        return $this;
    }

    /**
     * Set a row offset on the query
     *
     * @param int
     * @return object
     */
    public function skip($num) {
        $this->offset = $num;

        return $this;
    }

}
```