This document explains how new SomeRow's are created for new database table.

The idea behind the solution

Database table has rows. Each of these rows usually is the target for 4 operations:
Create
Read
Update
Delete

The first letters form acronym CRUD, that is used to refer this kind of group of actions.

Each database table is different, but *most* of them have the same properties: they all have 1) table name, 2) primary key and 3) column names. From these we can do pseudo code for example to create operation:

```
columns = getColumns;
table = getTable;
primary key = getPrimary;
```

SQL expression is formed:
INSERT INTO (names of the columns) VALUES(values of the columns)

When creating new row, new primary key id also created. Primary column must be of data type SERIAL (in mysql usually: int not null auto_increment).

All the other sql expressions are similar in a way that they use columns, table and primary key information to create INSERT, UPDATE, SELECT or DELETE sql expression.

update()

UPDATE table set column1=value1, column2=value2 WHERE primary key column=primary key value.

DELETE FROM table where primary key column = primary key value

Technical syntax things.

Objects members can be accessed using members name in string variable. For example if we want to access SomeRowUser's id member, following syntax works:

```
$primarykey = 'id';
$id = $someuserrow->$primarykey;
```

Because primary key name is fetched using getPrimary() following also works:
.. this code is in SomeRow, but it is extended by SomeRowUser, and SomeRowUser implements getPrimary() method. In the case of someuserrow, getPrimary() returns string id.

```
$primarykey = $this->getPrimary();
$usersid = $this->$primarykey.
```

The whole logic of CRUD operations can be coded by using only table, primarykey and column names as changing factor. That is why it is possible to place create(), read(), update() and delete() to superclass, and just demand those three methods to be in extending classes.

The walk through to create new somerow

Lets say, that one creates new database table called somequotes, with columns id, quote, author.

The table name is 'somequotes'.
Primary key name is 'id'.
Column names are 'id', 'quote' and 'author'.

Programmer must create new class that does not exist yet, SomeRowQuotes is good choice for name. Class suffix must be the same as filename in /library/database/rows/ folder, in this case quotes.php.

That class, SomeRowQuotes must extend from SomeRow. It MUST implement at least three methods, getPrimary(), getColumns() and getTable().

```
public function getPrimary() {
   return 'id'; //this column is "id INT SERIAL PRIMARY KEY"
}

public function getTable() {
   return "somequotes";
}

public function getColumns() {
   return array('id','quote','author'); //primary key must be first, order of others is not significant
}
```

That's it. In this simple case, all is done. The SomeRow in its crud methods will take care of the rest.

SomeRow is factory class, and it has factory method called getRow(name). The argument name must have naming convention such, that rows/ subfolder has fileof that name, and it has SomeRow[NAME] called class, example:
/library/some/database/row.php // has class SomeRow and factory method SomeRow
/library/some/database/rows/quotes.php //has class SomeRowQuotes extends SomeRow

Usage examples

...create new row

```
$quotesrow = SomeRow::getRow('quotes');
$quotesrow->id = null; //just to make it visible that it is null
$quotesrow->quote = 'The cat is greener that a black dog';
```

```
$quotesrow->author = 'Hannu Something';
$id = $quotesrow->create();

...read
$id = SomeRequest::getInt('quotesid',null);
$quotesrow = SomeRow::getRow('quotes');
$quotesrow->id = $id;
$didfindrow = $quotesrow->read();
echo $quotesrow->author;

...update
$id = SomeRequest::getInt('quotesid',null);
$quotesrow = SomeRow::getRow('quotes');
$quotesrow->id = $id;
$quotesrow->author = 'Some Lohtander'; //change author
$didupdate = $quotesrow->update();

...delete
$id = SomeRequest::getInt('quotesid',null);
$quotesrow = SomeRow::getRow('quotes');
$quotesrow->id = $id;
$diddelete = $quotesrow->delete();
```

Non standard database tables

Of course, not all databases have serial type key. Most obvious example is session table, where key is 32 characters long md5 hash (see md5(), www.php.net/md5).

In this case, the update, read and delete will work as expected, but SomeRow create() method relies on the serial and last insert id functionality.

Programmer must override create() method in SomeRowSession class. Then it can be used as all other SomeRow classes. Overriding create method means, that SomeRowSession create() expects that ID sesskey IS GIVEN, it must be present, and it uses that in INSERT sql expression. This is different from the create() in the SomeRow. It does not set primary key to INSERT, as the database creates it when it is not set (This is exactly what the serial is for in Postgres and auto_increment in MySQL).Implementation of create() in SomeRowSession is such, that it must check if row already exists, and then either insert or update that row.

When one wants SomeRow to do even more complex things as session row, other methods can be overridden too. Example, if for some reason this is where content language abstraction happens, then all the create() read() update() and delete() are overridden to add (WHERE) “language=$lang” to sql query. SomeRow still should be used, as it defines programmer the CRUD interface, so API is still the same.

Template method pattern, Factory pattern

There is a bit of a mind twist in the way SomeRow is done. This software pattern is called Template

Method. It means, that superclass SomeRow does most of the functionality itself, but in order to do it, it need child class to implement parts of required methods. This is almost the same as callback arguments in usort and similar functions (though that is closer to something called Strategy pattern...). In this case, the needed parts are column names, primary key name and table name information. So there are three template methods, that child class implements.

There is another pattern also here, called factory. Most of the novice programmer create new classes by extending some super class, and instantiate classes directly. For example like this:

```
someloader('some.database.rows.rowuser'); // file would need to be named rowuser if used directly
$user = new SomeRowUser();
```

It is possible to do it like that. It is just, that lots of code can be placed to factory classes factory method, and at the same time, the factory class can be the superclass.

This kind of coding style also pays off later, when things change with new requirements. Factory method is something playfully called “a single-point-of-failure”. It does not mean, that code must run into bugs, it means that when there is need for change for any reason, it is fixed in one place; in this case it is factory method.

These both patterns also help to create automated tests, so called Unit tests. Best practices are built on top of each others.