# classysql.php version 1.5

## *connect to a database*

```
include_once('classysql.php');
$config['y_db_name'] = 'thing_-_thing.com';
$config['y_db_user'] = 'whoever';
$config['y_db_password'] = 'whatever';
$db = new y_db($config);
```

## *simple select*

```
        SELECT * FROM `myrecord`
        WHERE `mykey`='123';
class myrecord {
  var $mykey;
  var $myvalue1;
  var $myvalue2;
  …
  // can be subset of fields in database
}
$r = new myrecord();
$r->mykey = 123;
switch ($db->select($r)) {
case 1: print_r($r); break; // OK
case 0: echo "record not found"; break;
default: echo "more than one record found"; break;
}
```

## *selecting more than one record*

```
        SELECT * FROM `myrecord`
        WHERE `mykey`='123';
$r = new myrecord();
$r->myvalue1 = 'occurs several times';
$q = $db->query();
while ($q->select($r) > 0) { print_r($r); }
```

## *alternatively, collecting results…*

```
$a = array();
while ($q->select($r) > 0) { $a[] = clone $r; }
// omit 'clone' in php4, essential in php5
```

## *counting records*

```
        SELECT COUNT(*) FROM `myrecord`
        WHERE `myvalue1`=' occurs several times';
$r = new myrecord();
$r->myvalue1 = 'occurs several times';
$n = $db->info($r, y_op::count());
echo "occurs {$n} times";
```

## *max value*

```
        SELECT MAX(`mykey`) FROM `myrecord`;
$r = new myrecord();
$max = $db->info($r, y_op::max('mykey'));
/* and similarly for other functions
such as min and sum */
```

## *explicit 'where' condition*

```
        SELECT * FROM `myrecord`
        WHERE `mykey`<' 123';
$r = new myrecord();
$q = $db->query();
$q->where(y_op::lt('mykey', 123));
while ($q->select($r) > 0) { print_r($r); }
```

## *combining conditions*

```
        SELECT COUNT(*) FROM `myrecord`
        WHERE `mykey`<' 123'
                AND 'myvalue1' > '99'
                AND 'myvalue2' = '42';
$q = $db->query();
$yop1 = y_op::lt('mykey', 123);
$yop2 = y_op::gt('myvalue1', 99);
$yop3 = y_op::eq('myvalue2', 42);
$yop = y_op::aand($yop1, $yop2, $yop3);
$q->where($yop);
while ($q->select($r) > 0) { print_r($r); }
```

## *alternatively…*

```
$a = array(y_op::lt('mykey', 123),
            y_op::gt('myvalue1', 99),
            y_op::eq('myvalue2', 42));
$q->where(y_op::aand($a));
```

## *inserting a record*

```
        INSERT `myrecord`
        SET `mykey`=42, `myvalue1`='a value';
$r = new myrecord();
$r->mykey = 42;
$r->myvalue1 = 'a value';
// not all fields need be set
$db->insert($r);
```

## *inserting multiple records*

```
        INSERT `myrecord`(`mykey`,`myvalue1`)
        VALUES (42, 'mva'), (43, 'mvb');
$r1 = new myrecord();
$r1->mykey = 42; $r1->myvalue1 = 'mva';
$r2 = new myrecord();
$r2->mykey = 43; $r2->myvalue1 = 'mvb';
$n = $db->insert($r1, $r2);
echo "{$n} records inserted";
```

## *alternatively…*

```
$a = array(new myrecord(), newmyrecord(),…);
// populate $a elements
$db->insert($a);
```

## *deleting record by field value*

```
        DELETE FROM `myrecord`
        WHERE `mykey`='42';
$r = new myrecord();
$r->mykey = 42;
$db->delete($r, 'mykey');
```

## *deleting records conditionally*

```
        DELETE FROM `myrecord`
        WHERE `myvalue1`<'123';
$r = new myrecord();
$q = $db->query();
$q->where(y_op::lt('myvalue1', 123));
$q->delete($r); // or just $q->delete('myrecord');
```

## *updating records*

```
        UPDATE `myrecord`
        SET 'myvalue' = 199 WHERE `mykey`='42';
$r = new myrecord();
$r->mykey = 42;
$db->select($r);
$r->myvalue1 = $r->myvalue1 + 99;
$db->update($r, 'mykey');
// will also update myvalue2 unless unset
```

## *updating multiple records*

```
        UPDATE `myrecord`
        SET 'myvalue2' = 'set several records to this'
        WHERE `myvalue1`=' when this appears';
$r = new myrecord();
$r->myvalue2 = 'set several records to this';
$r->myvalue1 = 'when this appears';
$db->update($r, 'myvalue1');
```

## *alternatively…*

```
$r = new myrecord();
$r->myvalue2 = 'set several records to this';
$q = $db->query();
$q->where(y_lt('myvalue1', 123));
$q->update($r, 'myvalue1');
```

## *ordering results*

```
        SELECT * FROM `myrecord`
        WHERE `myvalue1`<' 123'
        ORDER BY `myvalue1` ASC,
                'myvalue2` DESC;
$r = new myrecord();
$q = $db->query();
$q->where(y_lt('myvalue1', 123));
$q->ascending('myvalue1'); // primary sort
$q->descending('myvalue2'); // secondary sort
while ($q->select($r) > 0) { print_r($r); }
```

## *selecting distinct*

```
        SELECT DISTINCT * FROM `myrecord`
        WHERE `mykey`<' 123';
$r = new myrecord();
$q = $db->query();
$q->where(y_lt('myvalue1', 123));
$q->distinct();
while ($q->select($r) > 0) { print_r($r); }
```

## *a more useful select distinct*

```
        SELECT DISTINCT `myvalue2`
        FROM `myrecord`
        WHERE `mykey`<' 123';
$r = new myrecord();
$q = $db->query();
$q->where(y_lt('myvalue1', 123));
$q->distinct('myvalue2');
while ($q->select($r) > 0) { print_r($r); }
```

## *natural join*

```
        SELECT *
        FROM `myrecord` AS `a`,
                `otherrecord` AS `b`
        WHERE `a`.`mykey`=
                `b`.`myrecord_mykey'
           AND `a`.`myvalue1`=99;
class otherrecord {
  var $myrecord_mykey;
  var $otherfield1; …
}
$ra = new myrecord();
$rb = new otherrecord();
$joiner = array($ra, $rb);
$q = $db->query();
$ra->myvalue1 = 99;
while ($q->select($joiner) > 0) {
  print_r($joiner[0]); print_r($joiner[1]); }
```

## *alternatively…*

```
…
while($q->selectjoin($ra, $rb) > 0) {… }
/* just avoids the need to make up an array */
```

## *what if fields not named like that?*

```
class otherrecord {
  var $myotherkey;
  var $otherfield1; …
}
$ra = new myrecord();
$rb = new otherrecord();
$joiner = array($ra, $rb);
$q = $db->query();
$ra->myvalue1 = 99;
$q->where(y_op::feq('mykey', 'myotherkey'));
while ($q->select($joiner) > 0) {
  print_r($joiner[0]); print_r($joiner[1]); }
```

## *or, define equivalent fields*

[classysql assumes that a field in class a is equivalent to a field called f in class b is the field in a is called b_f. This function overrides that.]

```
class myrecord {
  var $mykey; …
  function y_xrefs($classname) {
    if ($classname == 'otherclass') {
      return array('mykey'=>'otherkey');
    } else { return NULL; }
  }
}
```

## *what if fields named the same?*

```
class otherrecord {
  var $mykey;
  var $otherfield1; …
}
$ra = new myrecord();
$rb = new otherrecord();
$joiner = array($ra, $rb);
$q = $db->query();
$ra->myvalue1 = 99;
$q->where(y_op::feq(
  y_op::field('mykey', 0),
  y_op::field('mykey', 1));
/* the 0 and 1 are indexes of $ra and $rb
    in $joiner, respectively; a y_op_field can be given
    pretty much anywhere that a field name is
    needed */
while ($q->select($joiner) > 0) {
  print_r($joiner[0]); print_r($joiner[1]); }
```

## *a (three-way) self join*

```
SELECT DISTINCT `a`.`mykey`
FROM `myrecord` AS `t0`,
       `myrecord` AS `t1`,
       `myrecord` AS `t2`,
WHERE `t0`.`myvalue1`='42'
   AND `t1`.`myvalue1' ='99'
   AND `t2`.`myvalue1`='12';
   AND `t0`.`mykey`=`t2`.`mykey`
   AND `t1`.`mykey`=`t2`.`mykey`;
```

[Aside: why would I do a join like this? – Consider an index of words vs. keys of another table. If the query is to "find records containing aaa and bbb and ccc", then a 3-way self-join on distinct key will do this]

```
$ra = new myrecord(); $ra->myvalue1 = 42;
$rb = new myrecord(); $rb->myvalue1 = 99;
$rc = new myrecord(); $rc->myvalue1 = 12;
$joiner = array($ra, $rb, $rc);
$ks = array(y_op::field('mykey',0),
            y_op::field('mykey',1),
            y_op::field('mykey',2));
$q = $db->query();
$q->distinct($ks[0]);
$q->where(y_op::aand(
   y_op::feq($ks[0], $ks[1]),
   y_op::feq($ks[1], $ks[2])
));
while ($q->select($joiner) > 0) { … }
```

## *similarly, count self join results*

```
…
$q = $db->query();
$q->where(y_op::aand(
   y_op::feq($ks[0], $ks[1]),
   y_op::feq($ks[1], $ks[2])
));
$n = $q->info(y_op::count_distinct($ks[0]));
```

## *class differs from table name*

```
class myrecord {
  var $mykey;
  var $myvalue1;
  var $myvalue2;
  function y_table() { return 'thetable'; }
}
```

## *but, if all you want is a common table name prefix…*

```
$config['y_db_prefix'] = 'myprefix_';
$db = new y_db($config);
```

## *what if my class properties aren't named same as database fields?*

```
class myrecord {
  var $mykey;
  var $myvalue1;
  var $myvalue2;
  function y_fields() {
    static $fa = array('mykey'=>'thekey',
                       'myvalue1' => 'thev1',
                       'myvalue2' => 'thev2');
    return $fa;
  }
}
```