SELECT

```
db_select($table, $alias = NULL, $options = array())
```

$table	Database table to select from.
$alias	Table alias.
return	New query object.

```
$query = db_select('users', 'u')
  ->fields('u',
    array('uid', 'name'));
$result = $query->execute();
```

```
->distinct($distinct = TRUE)
```

$distinct	Flag indicating DISTINCT query.
return	The called query object.

```
->fields($table_alias, $fields = array())
```

$table_alias	Alias of the table the field belongs to.
$fields	Array of field names.
return	The called query object.

```
->addField($table_alias, $field, $alias = NULL)
```

$table_alias	Alias of the table the field belongs to.
$field	Field name.
$alias	Field alias.
return	Unique field alias.

```
->range($start = NULL, $length = NULL)
```

$start	First record of the result set.
$length	Max number of records.
return	The called query object.

```
->groupBy($field)
```

$field	The field to group by.
return	The called query object.

```
->orderBy($field, $direction = 'ASC')
```

$field	The field to order by.
$direction	'ASC' or 'DESC'.
return	The called query object.

```
->orderRandom()
```

return	The called query object.

```
->union(SelectQueryInterface $query, $type = '')
```

$query	Query to union.
$type	Type of union.
return	New resulting query object.

```
->addExpression($expression, $alias = NULL, $arguments = array())
```

$expression	Expression string.
$alias	Expression alias.
$arguments	Assoc array of placehoders and placeholder values.
return	Unique expression alias.

```
->countQuery()
```

return	New query object.

```
->addTag($tag)
```

$tag	Query identification.
return	The called query object.

```
->hasTag($tag)
```

$tag	Query identification.
return	TRUE if condition is met.

CONDITIONS

```
->condition($field, $value = NULL, $operator = NULL)
```

$field	The field to check or the result of a logic operation (or, and, xor)
$value	The value to test against.
$operator	Default: '=' or 'IN'. Supported: =, <, >, >=, <=, IN, NOT IN, LIKE, BETWEEN, IS NULL, IS NOT NULL
return	The called query object.

```
->where($snippet, $args = array())
```

$snippet	Where clause (with placeholders)
$args	Assoc array of placeholders and placeholder values.

```
->db_or()->condition()->condition()
```

return	Condition of OR-ed conditions.

```
->db_and()->condition()->condition()
```

return	Condition of AND-ed conditions.

```
->isNull($field)
->isNotNull($field)
```

$field	The field to check.
return	The called query object.

```
->exists(SelectQueryInterface $select);
->notExists(SelectQueryInterface $select);
```

$select	The query to check.
return	The called query object.

JOIN

```
->join($table, $alias = NULL, $condition = NULL, $arguments = array())
```

$table	The table to join with.
$alias	Table alias.
$condition	Join conditions.
$arguments	Assoc array of placeholders and placeholder values.
return	Unique table alias.

```
$query = db_select('users', 'u');
$query->innerJoin('node', 'n',
  'n.uid = u.uid');
$query->addField('u', 'name');
$query->addField('n', 'title');
$result = $query->execute();
```

```
->innerJoin ($table, $alias = NULL, $condition = NULL, $arguments = array())
->leftJoin ($table, $alias = NULL, $condition = NULL, $arguments = array())
->rightJoin ($table, $alias = NULL, $condition = NULL, $arguments = array())
```

See *join* method.

wizzlern
de Drupal trainers

PAGER

```
->extend('PagerDefault')
```

return	New pager extender object.

```
->extend('PagerDefault')->limit($count)
```

$count	Number of items per page.

SORTABLE TABLE

```
->extend('TableSort')
```

return	Table extender object.
return	The called query object.

```
->extend('TableSort')->orderByHeader($header)
```

$header	Array with sorting criteria.
return	The called query object.

```
$header = array(
    array(
    'data' => t('Title'),
    'field' => 'n.title',
    'sort' => 'desc'),
  t('Operations'),
);
```

RESULTS

```
->execute($args = array(), $options
= array())
```

return The called query object.

```
->fetch($mode = NULL,
$cursor_orientation = NULL,
$cursor_offset = NULL)
```

$mode Fetch mode.
return Result type specified by $mode.

```
->fetchObject($class_name = NULL,
$constructor_args = array())
```

$class_name Class type to be returned.
Default: stdClass
return Object of one record.

```
->fetchAssoc()
```

return Associative array of one record.

```
->fetchAllAssoc($key, $fetch = NULL)
```

$key Field name of the array key
$fetch Fetch mode
(PDO::FETCH_ASSOC,
PDO::FETCH_NUM, or
PDO::FETCH_BOTH).
return Associative array of data objects

```
->fetchAll($mode = NULL,
$column_index = NULL,
$constructor_arguments = array())
```

$mode Fetch mode. See above.
return Array of data objects. Depending on fetch mode.

```
->fetchField($index = 0)
```

$index Numeric index of the column.
return A single field.

```
->fetchAllKeyed($key_index = 0,
$value_index = 1)
```

$key_index Numeric index of the array key.
$value_index Numeric index of the array value.
return Associative array of all records.

```
->fetchCol($index = 0)
```

$index Numeric index of the column.
return Array of all records.

INSERT

```
db_insert($table, $options = array
())
```

$table Database table to insert into.
return New query object.

```
$nid = db_insert('node')
  ->fields(array(
    'title' => 'Example',
    'uid' => 1,
    'created' => REQUEST_TIME))
  ->execute();
```

```
->values(array $values)
```

$values Assoc array of values to insert.
return The called query object.

```
$nid = db_insert('node')
  ->fields(array('title', 'uid',
    'created'))
  ->values(array(
    'title' => 'Example',
    'uid' => 1,
    'created' => REQUEST_TIME))
  ->execute();
```

```
->from(SelectQueryInterface $query)
```

$query Select query to fetch the rows that should be inserted.
return The called query object.

UPDATE

```
db_update($table, $options = array
())
```

$table Database table to update.
return New query object.

```
$num_updated = db_update('node')
  ->fields(array(
    'uid' => 5,
    'status' => 1,
    ))
  ->condition('created',
    REQUEST_TIME - 3600, '>=')
  ->execute();
```

MERGE

```
db_merge($table, $options = array())
```

$table Database table to merge into
return New query object

```
db_merge('role')
  ->key(array('name' => $name))
  ->fields(array(
      'weight' => $weight,
      ))
  ->execute();
```

```
->key(array $fields, $values = array
())
```

$fields Array of fields to match or set.
Or associative array of fields and values.
$values Values to set.
return The called query object.

DELETE

```
db_delete($table, $options = array
())
```

$table Database table to delete from.
return New query object.

```
$num_deleted = db_delete('node')
  ->condition('nid', 5)
  ->execute();
```

TRUNCATE

```
db_truncate($table, $options = array
())
```

$table Database table to remove.
return New query object.

wizzlern
de Drupal trainers

QUERIES

```
db_query($query, $args = array(),
$options = array())
```

Note: Access control is not supported! Query may not be compatible with other database types.

settings.php

Single database configuration example:

```
$databases['default']['default'] =
  array(
    'driver' => 'mysql',
    'database' => 'databasename',
    'username' => 'username',
    'password' => 'password',
    'host' => 'localhost',
    'prefix' => '',
    'collation' =>'utf8_general_ci',
  );
```

DEBUGGING

```
print($query->__toString());
```

DOCUMENTATION

Database API on drupal.org:
http://drupal.org/developing/api/database

jQuery
write less, do more.

PHP

drupal_add_js($data = NULL, $options = NULL)

$data	Depending on option 'type':
file	Relative path to file
inline	Javascript code
external	Absolute path to file
setting	Array of settings
$options	Type or array of options:
type	*file*, inline, external, setting
scope	*header*, footer
group	Group: JS_LIBRARY, JS_DEFAULT, JS_THEME
every_page	TRUE = Load on all pages.
weight	Sort order
defer	<script> defer attribute.
cache	*TRUE* = Cache this Javascript.
preprocess	JS aggregation enabled.

```
drupal_add_js('misc/collapse.js');

drupal_add_js(array('myModule' => array
('my_setting' => 'my value')), 'setting');

drupal_add_js(
  'jQuery(document).ready( function () {
    alert("Hello!");
  });',
  array('type' => 'inline',
    'scope' => 'footer', 'weight' => 5,
  ),
);

drupal_add_js('http://example.com/example.js',
'external');
```

hook_js_alter(&$javascript)

$javascript	Array of JavaScript (files) to be added to the page.

JQUERY

Compatibility code

```
(function ($) {
  // ... Your code here ...
}(jQuery));
```

Execute code when DOM is loaded

```
$(function() {
  // ... Your code here ...
});
```

Drupal.settings

```
<?php
  drupal_add_js(array(
    'my_module' => array('color' => 'green')),
    'setting'
  );
?>

$('#id')
 .css({'color': Drupal.settings.my_module.color});
```

Drupal.behaviors

```
Drupal.behaviors.mybehavior = {
  attach: function (context, settings) {
    $('#id', context)
      .once('mybehavior', function () {
      // ... Your code here ...
    });
  }
};
```

Drupal.checkPlain(str)

str	String to be sanitised.
return	Sanitised string

Drupal.t(str, args)

str	String to be translated.
args	Replacement arguments.
!arg	Unchanged
@arg	Sanitized
%arg	Sanitized + em tag
return	Translated string.

```
Drupal.t('@lang string', {'@lang': language});
```

wizzlern
de Drupal trainers

Drupal.formatPlural(count, singular, plural, args)

count	Number to be included in string.
singular	Singular string
plural	Plural string
args	Replacement arguments. See Drupal.t()
return	Translated string.

```
Drupal.formatPlural(count, 'One string', '@count
strings');
```

Drupal.theme.prototype

```
Drupal.theme.prototype.placeholder = function
(str) {
  return '<em class="placeholder">'
    + Drupal.checkPlain(str) + '</em>';
};
```

Drupal.theme(func, args)

func	Theme function.
args	Arguments for theme function.
return	HTML output.

```
Drupal.theme('placeholder', 'Hello!');
```

Drupal.attachBehaviors(context, settings)

context	Context for the attach() method. Default: 'document'
settings	Settings for the attach() method. Default: Drupal.settings

STATES

#states

<state> State to be applied when conditions are met. See drupal_process_states().

For all form element types:
enabled, disabled, required. optional,
visible, invisible, checked, unchecked,
expanded, collapsed.

For some form element types:
relevant, irrelevant, valid, invalid,
touched, untouched, readwrite, readonly.

```
'#states' => array(
  // Show the form element if 'bar' has been
selected for 'foo'.
  'visible' => array(
    ':input[name="foo"]' => array('value' =>
'bar'),
  ),
),
```

AJAX

#ajax

callback	Callback function which can return HTML, renderable array, or array of AJAX Commands.
effect	Effect used when adding content *none*, fade, slide
event	Event at which the HTTP request occurs. Optional.
keypress	TRUE: Event is triggered when Enter key is pressed.
method	Manipulation method of the 'wrapper' content. *replace*, after, append, before, prepend.
path	Callback path. Default: system/ajax
prevent	Event(s) that should not trigger the ajax call.
progress	Progress bar settings: 'type', 'message', 'url', 'interval'.
trigger_as	Form element to handle the event trigger.
wrapper	Id attribute of DOM element to be replaced by callback. (no '#')

```
function poll_form(&$node, $form_state) {
  ...
  $form['choice_wrapper']['poll_more'] = array(
    '#type' => 'submit',
    '#value' => t('More choices'),
    '#description' => t("Add more choices."),
    '#weight' => 1,
    '#submit' =>
      array('poll_more_choices_submit'),
    '#ajax' => array(
      'callback' => 'poll_choice_js',
      'wrapper' => 'poll-choices',
      'method' => 'replace',
      'effect' => 'fade',
    ),
  );
  ...
}

function poll_choice_js($form, $form_state) {
  return $form['choice_wrapper']['choice'];
}
```

jQuery.ajax()

```
$.ajax({
  url: Drupal.settings.basePath
    + 'jquery_demo/ajax/' + string,
  success: function(data) {
    alert(data.message);
  }
  error: function (xmlhttp) {
    alert(Drupal.ajaxError(xmlhttp, db.uri));
  }
});
```

DRUPAL JS LIBRARIES

drupal_add_library($module, $name, $every_page = NULL)

$module	Module that registered a library.
$name	Library name.
$every_page	TRUE: add to every page.
return	TRUE on succes.

Drupal system libraries:
drupal.ajax, drupal.batch, drupal.progress, drupal.form, drupal.states, drupal.collapse, drupal.textarea, drupal.autocomplete.

jQuery libraries:
jquery, jquery.once, jquery.form, jquery.bbq, drupal.vertical-tabs, farbtastic, jquery.cookie.

jQuery UI libraries:
ui, ui.accordion, ui.autocomplete, ui.button, ui.datepicker, ui.dialog, ui.draggable, ui.droppable, ui.mouse, ui.position, ui.progressbar, ui.resizable, ui.selectable, ui.slider, ui.sortable, ui.tabs, ui.widget.

wizzlern
de Drupal trainers

jQuery UI Effects libraries:
effects, effects.blind, effects.bounce, effects.clip, effects.drop, effects.explode, effects.fade, effects.fold, effects.highlight, effects.pulsate, effects.scale, effects.shake, effects.slide, effects.transfer.

Other libraries:
once, form, jquery-bbq, vertical-tabs, cookie.

hook_library()

```
$libraries['ui.tabs'] = array(
    'title' => 'jQuery UI: Tabs',
    'website' => 'http://jqueryui.com/demos/tabs/',
    'version' => '1.8.7',
    'js' => array(
      'misc/ui/jquery.ui.tabs.min.js' => array(),
    ),
    'css' => array(
      'misc/ui/jquery.ui.tabs.css' => array(),
    ),
    'dependencies' => array(
      array('system', 'ui.widget'),
    ),
  );
```

DOCUMENTATION

jQuery: jquery.com, visualjquery.com, jqueryui.com
Drupal jQuery: Javascript startup guide