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