Database API

Your new friend

Francisco Alonso Borragán

DrupalCamp Spain 2014

About me

Francisco Alonso Borragán

Drupal Developer

www.kikoalonsob.com

IRC #drupal_es : kikoalonsob

Twitter : @kikoalonsob

Index

1. Intro
2. Static queries
3. Dynamic queries
    1. *Select queries*
    2. *Insert queries*
    3. *Update queries*
    4. *Delete queries*
4. Query alteration
5. Working against other DB
6. Questions

Intro

Abstraction layer for accessing database servers

- Support multiple database servers
- Dynamic construction of queries
- Enforce security checks and good practices
- Provide an interface for intercept and modify queries

Static queries

Static queries

- Explicit query string (Better for performance)
- Used for simple select queries
- Returns a prepared statement object, **already executed**

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

Static queries

```
$uid = 1;
$result = db_query('SELECT n.nid, n.title, n.created
        FROM {node} n WHERE n.uid = :uid',
        array(':uid' => $uid),
        array('fetch' => PDO::FETCH_ASSOC)
        );
```

Prefixing

All table names must be wrapped in **{ }**

Static queries

```
$uid = 1;
$result = db_query('SELECT n.nid, n.title, n.created
        FROM {node} n WHERE n.uid = :uid',
        array(':uid' => $uid),
        array('fetch' => PDO::FETCH_ASSOC)
        );
```

Placeholders

Literal or an array that will be inserted into a query for execution

Static queries

```
$uid = 1;
$result = db_query('SELECT n.nid, n.title, n.created
        FROM {node} n WHERE n.uid = :uid',
        array(':uid' => $uid),
        array('fetch' => PDO::FETCH_ASSOC)
        );
```

Query options

Set the query behavior (typically only two directives)

- **target** (master,slave)
- **fetch** (PDO::FETCH_OBJ, PDO::FETCH_ASSOC, etc)

Static queries

```
$uid = 1;
$result = db_query('SELECT n.nid, n.title, n.created
        FROM {node} n WHERE n.uid = :uid',
        array(':uid' => $uid),
        array('fetch' => PDO::FETCH_ASSOC)
        );
```

and...

how to use the result?

Iterating over the result

with a **foreach** loop...

```
foreach ($result as $record) {
// Do something with each $record
}
```

Iterating over the result

...or with a **while** loop

```
if ($result) {
        while ($record= $result->fetch()) {
        // Do something with each $record
        }
}
```

Instead of use `$result->fetch()`, we can use:

```
$result->fetchAssoc() //to fetch an associative array
$result->fetchObject() //to fetch an object
$result->fetchField($column_index) //to fetch just a single field
```

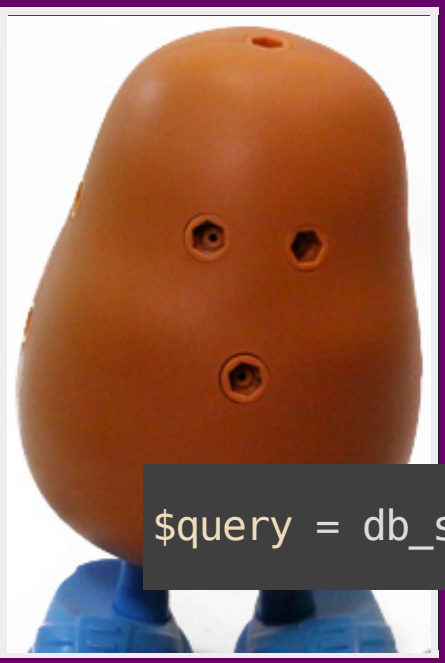
All results at once into an array

```
//Indexed array
$result->fetchAll()
//Associative array keyed by $field value
$result->fetchAllAssoc($field)
//associative array of field 1 => field 2
$result->fetchAllKeyed()
//associative array of field I => field j
$result->fetchAllKeyed(i, j)
//Single array
$result->fetchCol(i=0)
```

Dynamic queries

Dynamic queries

- Dynamically built
- Used for some select queries
- Used for all insert, update, delete and merge queries



Select queries

Always start using

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

and need at least one field

```
$query -> fields('table_or_alias', array('field_1','field_2'));
$query -> fields('table_or_alias'); //All table fields
```

Joins

You can add one or more joins

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

* `join` can be replaced by `innerJoin`, `rightJoin` or `leftJoin` too.

Conditional clauses

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

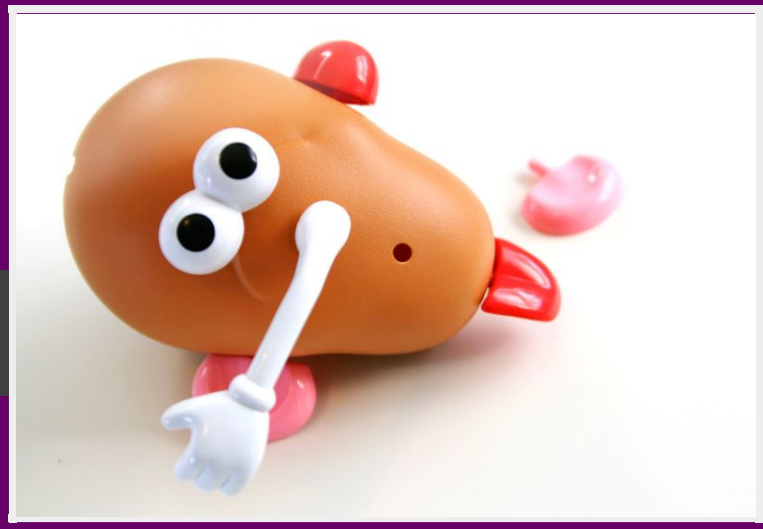
`$operator` accepts =, <, >=, `IN`, `LIKE`, `BETWEEN`, etc…

By default, conditions are concatenated with `AND`

Use `db_and()`, `db_or()` or `db_xor()` to overwrite it

```
db_or = db_or();
$db_or -> condition('type', 'page', '=');
$db_or -> condition('field_1', array(12, 15), 'IN');
$query -> condition ($db_or);
```

Ordering



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

Grouping

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

Ranges and limits

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

how to use the result?

Remember: `db_query()` returns a prepared statement object, already executed

But ... `db_select()` returns a SelectQuery object

```
$result = $query -> execute()
```

Now, you can use the `$result` like in static queries

Insert queries

Always start using:

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

We have to specify values with `fields($value)` function

- **Compact form:** The preferred form for most Insert queries
- **Degenerate form:** Useful for running a multi-insert query

Compact form

A simple $key => $value array where $key is the field name.

```
$query = ->fields(array(
        'title' => 'Example',
        'uid' => 1,
        )
)
$query->execute();
```

Degenerate form

- `fields()` only specifies field names
- `values()` specifies a `$key => $value` array

```
$query = db_insert('node');
$query -> fields(array('title', 'uid'));
$values = array(
        array('title' => 'Example', 'uid' => 1),
        array('title' => 'Example2','uid' => 1)
);
foreach ($values as $record) {
        $query -> values($record);
}
$query->execute();
```

Insert based on the result of a select

Build the select query, **but not execute!!**

```
$query = db_insert('node');
         -> from($select);
$query -> execute();
```

Update queries

Similar to insert queries, start using

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

and continuate specifying fields (like in insert queries) and conditions

```
$query -> fields($table_alias, array $fields = array())
       -> condition($field, $value, $operator);
$query -> execute();
```

Delete queries

Yes... start using

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

and continue with a condition

```
$query -> condition($field, $value, $operator)
       ->execute();
```

Query alteration

Most interesting feature of this API (IMHO)

Allow other modules to alter queries on the fly

Preparing your query

Add aone or more tags to identify the query

```
$query -> addTag('myTag');
```

Attach meta data to provide additional context (Opt.)

```
$query -> addMetaData('node', $node);
```

How to modify a query?

In your custom module, you can use

```
hook_query_alter(QueryAlterableInterface $query)
        // or
hook_query_TAG_alter(QueryAlterableInterface $query)
```

and inside them...

```
$query -> hasTag($tag);
$query -> hasAnyTag($tag_1, $tag_2);
$query -> hasAllTags($tag_1, $tag_2);
$query -> getMetaData($key);
```

Can I modify a views query?

Yes, you can!

Working against other DB

In your **settings.php**:

```
$databases = array();
$databases['default']['default'] = array(
  // Drupal's default credentials here.
);
$databases['my_other_db']['default'] = array(
  // Your secondary database's credentials here.
);
```

Working against other DB

In your module

```
db_set_active('my_other_db');
db_set_active(); //Connect to default
```

Working against other DB

You can define the connection directly in your module

```
$my_other_db = array (
        'database' => 'my_other_db',
        'username' => 'username',
        'password' => 'password',
        'host' => 'localhost',
        'driver' => 'mysql',
);
Database::addConnectionInfo('YourDatabaseKey', 'default', $my_other_db);
db_set_active('YourDatabaseKey');
```


QUESTIONS?
memegenerator.net

thanks