Database Abstractions: A Core Library

By: Brian Rosenthal and Zvi Boshernitzan

I. Overview

Associative Arrays, or dictionaries, are everywhere in SQL from the REQUEST variable to SMARTY template variable parameters, because name value pairs are a useful way to structure information.

Most SQL statements involve name value pairs, but traditionally, many SQL programmers treat SQL statements as text strings that they build up and edit.

This article describes an interface for abstracting database calls that is easy to implement in your application, that uses dictionaries natively, and thus, can make your database-driven application easier to implement with less code.

II. “Where Specifications”: db::where(...)

We define a “where spec” is an associatve array which specifies conditions on a SQL statement.

For example, consider the following dictionary:
{‘author_id’:1, ‘last_name’:’Rosenthal’}

This is the “where_spec” for:
‘author_id = 1 and last_name = ‘Rosenthal’

For the sake of simplifying common use-cases, we allow the following alternative syntaxes:

1. Just a number (assumes you mean “id”)
db::where(n) to be ‘id = 3’.

2. Just a string (assume you mean a SQL condition)
db::where(“last_name like ‘Smith%’ ”)

3. A sequence of conditions:
db::where([‘id > 5’, ‘id < 10’])

III. Creating and Altering Data

1. db::insert($table_name, $dictionary)

Inserts an entity into the database. Returns the id of the new entity.

Example:
db::insert(‘authors’, {‘first_name’:’Brian’, ‘last_name’:’Rosenthal’})

2. db::update($table_name, $dictionary, $where_spec)

Updates the entities in the specified table with the dictionary, matching the conditions in the where_spec.

Example:
db::update(‘authors’, {‘first_name’:’Brian’}, 3)
“Updates the first name of the author with id = 3 to be ‘Brian’”

db::update(‘authors’, {‘first_name’:’Brian’}, {‘last_name’:’Rosenthal’})
“Updates the first name of the authors with the last_name as ‘Rosenthal’ to be ‘Brian’”

3. db::delete($table_name, $where_spec)

Deletes entities from the specified table matching the conditions in the where-spec.

Example:
db::delete(‘authors’, 3)
“Deletes author with id = 3”

db::delete(‘authors’, {‘last_name’:’Rosenthal’})
“Deletes all authors with last-name Rosenthal.

4. db::update_or_insert($table_name, $dictionary, $where_spec)

Tries to update the entries as above, with the matching where_spec condition, and inserts the entry on failure.

IV. Queries that do not return a column

Parameters: $table_name, $where_spec

1. db::count($table_name, $where_spec)

Returns the number of rows in the specified table matching the conditions in the where_spec

Example:
How many books are there with author_id = 1?
db::count(‘books’, {‘author_id’:1})

2. db::exists($table_name, $where_spec)

Returns true if there exists a row in the specified table matching the where_spec. Otherwise, returns false.

Example:
Are there any books with author_id = 1?
db::exists(‘books’, {‘author_id’:1})

V. Queries that return one column

Parameters: (a) sql OR (b) $table_name, $column_name, $where_spec

1. db::string(...)

Returns the first field from the first row of the query, or null if there are no rows.

Example:
db::string(‘select title from books where id = 3’)
db::string(‘books’, ‘title’, 3)

2. db::strings(...)

Returns a sequence of strings, the first field of each of the result-set.

Example:
db::strings(“select id from customers where last_name like ‘Smith%’ ”);
db::strings(‘customers’, ‘id’, “last_name like ‘Smith%’ ”);

3. db::sum(...)

The sum of the specified column, across rows that match the condition.

Example:

db::sum(‘line_items’, ‘total’, ‘sales_order_id = 3’)

VI. Queries that return more than one column

Parameters: (a) SQL or (b) table_name, where_spec

1. db::object, db::assoc, db::row

Returns a structure containing the first row of the result-set.

Examples:

“Get customer id 3 as an object”
db::object(‘select * from customers where id = 3’);
db::object(‘customers’, 3)

... OBJECT:{‘first_name’:’Zvi’, ‘last_name’:’Boshernitzan’}

“Get customer id 3 as an associative array”
db::assoc(‘customers’, 3)

...{‘first_name’:’Zvi’, ‘last_name’:’Boshernitzan’}

“Get customer id 3 as an sequence of fields”
db::row(‘customers’, 3)
...[3, ‘Zvi’, ’Boshernitzan’]

2. db::objects, db::assocs, db::rows

Returns a sequence of structures containing the rows of the result set.

Example:
“Get a sequence of sales orders for customer id 3”
db::assocs(‘select * from sales_orders where customer_id = 3’)