Database Access

Tiki has a database query layer to handle queries affecting a single table. The layer builds the SQL based on arguments rather than having to concatenate strings. The resulting code is cleaner. SQL will still be used in Tiki libraries, but should be reserved for cases where relational logic is needed or advanced features provide a significant benefit.

Complex listings with multiple filters that are targeted towards end-users should use Unified Index instead of direct SQL.

The database layer works at the table-level, but does not perform any validation on the fields. It is only provided as a convenience.

The table class can be obtained through TikiDb and TikiDb_Bridge instances through the *table($tableName)* method.

Insert a row

Inserting records is a common task and the SQL syntax requires enumerating the fields and then the values, leading to common mistakes with missing arguments. With parameter bindings, all those values are question marks that must be synchronized with bind variables. The table access class allows to provide an array containing key-value pairs and will build the SQL accordingly.

Insert a record

```
<?php $pages = $tikilib->table('tiki_pages'); $id = $pages->insert(array( 'pageName' => 'HelloWorld', 'data' => 'Content here', // More fields here... )); // $id contains the auto-increment ID.
```

Deleting records

The table class does not abstract much from SQL on purpose. It only provides facilities to be more explicit about the task to be performed. Deleting records requires to build conditions to match the records. Just like insert queries, those conditions are provided as arrays.

Delete a record

```
<?php $pages = $tikilib->table('tiki_pages'); $pages->delete(array( 'page_id' => 42, ));
```

By default, update and delete queries built will use LIMIT 1 to avoid excessive damage from being caused by bad queries. When multiple records need to be deleted, *deleteMultiple(array $conditions)* can be used to remove the limitation.

Multiple conditions can be provided resulting in **AND** matching. For conditions that require different operators than equality, various expressions can be used. Expressions are described in detail further down as they are shared between different methods.

Delete a record

```
<?php $sessions = $tikilib->table('tiki_sessions'); $sessions->deleteMultiple(array( 'expiry' => $sessions->lesserThan(time()), 'session_id' => session_id(), ));
```

In the above, the generated condition will be *`expiry` < 1234567890 AND `session_id`* = "*1234567890*", or something equivalent.

Updating records

To update records, the data to be updated and the conditions matching the rows must be provided. Just like for delete, update will have a limitaiton on one updated record by default. *updateMultiple(array $data, array $conditions)* can be used instead.

Some expressions are also provided to perform non-direct assignments.

Delete a record

```
<?php $pages = $tikilib->table('tiki_pages'); $pages->update(array( 'last_visit' => 'foobar', 'hits' => $pages->increment(1), ), array( 'page_id' => 42, ));
```

In the above, the *hits* field will be set to *`hits` = `hits` + 1* while the other field will simply be assigned a value.

Retrieve data

Depending on what information is desired, multiple options are available. The complete function granting access to all functionality is *fetchAll(...)*. However, some common situations can benefit from a more compact form. The available methods are:

- **fetchBool(array $conditions)** : Returns true on a match, or false otherwise. Perfect as a lightweight query or for use in a control statement
- **fetchCount(array $conditions)** : Provides the result count only
- **fetchOne($field, array $conditions)** : Provides a single value coming from one record
- **fetchColumn($field, array $conditions, $maxRecords = -1, $offset = -1, $sort = null)** : Provides all the matched values from a single column
- **fetchMap($keyField, $valueField, array $conditions, $maxRecords = -1, $offset = -1, $sort = null)** : Retrieves the two values from the table and generates a map from the key and the value
- **fetchRow(array $fields, array $conditions)** : Retrieve the selected fields from a single row
- **fetchFullRow(array $conditions)** : Retrieve all fields from a single row
- **fetchAll(array $fields, array $conditions, $maxRecords = -1, $offset = -1, $sort = null)** : Fully-customizable fetch providing an array of associative arrays.

Common pagination case

```
<?php $pages = $this->table('tiki_pages'); $conditions = array( 'pageName' => $pages->like('User%'), ); $result = $pages->fetchAll( array('pageName', 'description', 'hits'), $conditions, $maxRecords, $offset, array('pageName' => 'ASC')); $cant = $pages->fetchCount($conditions);
```

The sort argument can be provided as an array containing multiple sort options or a generic expression.

The field lists can contain strings for field names or expressions (see below). The *all()* expression can also be used instead of an array for *SELECT **.

Expressions

The table class will use strings and scalar values to perform common operations like assignation and equality conditions. However, many situations step outside those boundaries. For those, expressions are provided. Essentially, expressions are SQL fragments with bound variables.

- **expr($fragment, array $arguments = array())**

    - Most generic usage, allows to insert SQL in many places.
    - In update for the data, they are used for the values.
    - In conditions, they represent the whole condition.
    - In a select query, they represent a single field.
    - An expression can be used instead of the sort array to replace the entire *order by* argument.
    - Within the fragment, *$$* will be replaced by the field for conditions.
    - All other expressions are just shorthands for this one.
- Condition expressions
    - **lesserThan($value)**
    - **greaterThan($value)**
    - **like($value)**
    - **unlike($value)**
    - **contains($value)**
    - **not($value)**
    - **exactly($value)** (binary safe compare)
    - **in(array $values)**
    - **between(array $values)** Must pass 2 values. Will match the values and the range between them.
- Field expressions
    - **count()**
    - **sum($field)**
    - **max($field)**
    - **all()** (for all fields, not a specific field, returns an array of expressions)
- Update value expressions
    - **increment($count)**
    - **decrement($count)**

Example of expressions: in() & sum()

<?php $files = $this->table('tiki_files'); $conditions = array(); if (! empty($galleryId)) { $galleryIds = array(); $this->getGalleryIds($galleryIds, $galleryId, 'list'); $conditions['galleryId'] = $files->in($galleryIds); } return $files->fetchOne($files->sum('filesize'), $conditions);