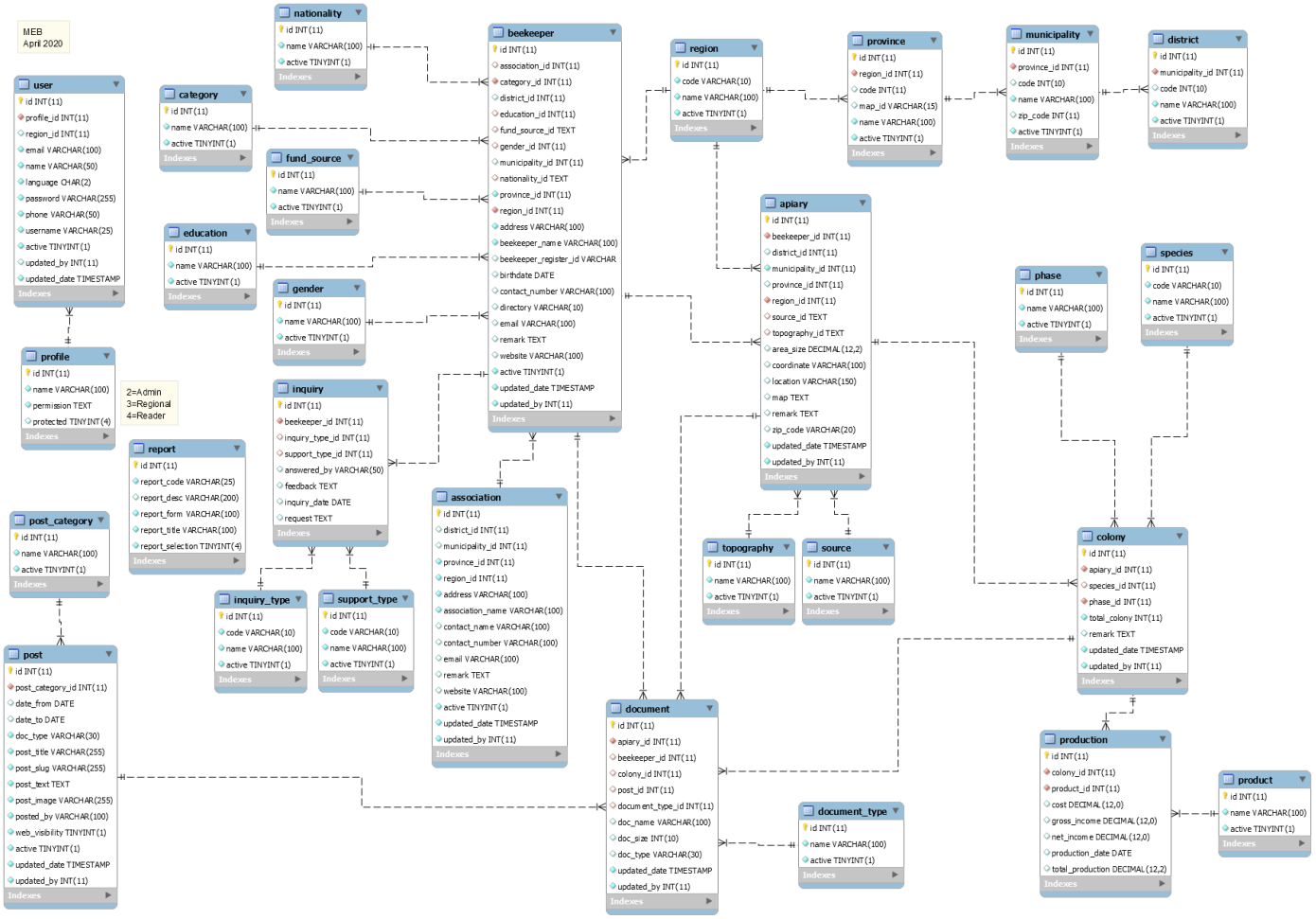
Database

MySQL is the world's most popular open source database. With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice. Oracle drives MySQL innovation, delivering new capabilities to power next generation web, cloud, mobile and embedded applications. MySQL is a relational database, in that it allows tables to be joined together and also supports the concept of foreign keys.

The database has several relationships between the tables, which is illustrated by the following diagram. It's possible to relate the tables directly in MySQL but the technology choice for the database is to JOIN the tables when it's needed in the model. The performance is better and the management of the database will be easier.

Database Diagram

MySQL Workbench is used to create the diagram of the database. The relations are made manually after a synchronization with the database.



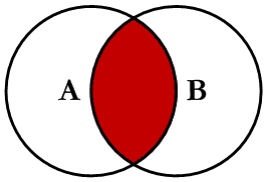
Relations between tables

The type of JOIN will determine which data will be selected. There is many different ways you can return data from two relational tables. I am excluding cross Joins and self referencing Joins. For the database most of the time, only 3 JOIN methods have been used (JOIN – LEFT JOIN – RIGHT JOIN).

1. INNER JOIN OR JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. OUTER JOIN
5. LEFT JOIN EXCLUDING INNER JOIN
6. RIGHT JOIN EXCLUDING INNER JOIN

7. OUTER JOIN EXCLUDING INNER JOIN

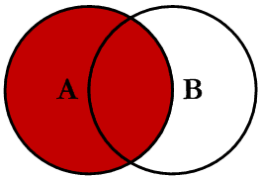
JOIN (or INNER JOIN)



This is the simplest, most understood Join and is the most common. This query will return all of the records in the left table (table A) that have a matching record in the right table (table B). This Join is written as follows:

SELECT <select_list>
FROM Table_A A
JOIN Table_B B
ON A.Key = B.Key

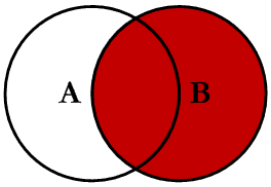
LEFT JOIN



This query will return all of the records in the left table (table A) regardless if any of those records have a match in the right table (table B). It will also return any matching records from the right table. This Join is written as follows:

SELECT <select_list>
FROM Table_A A
LEFT JOIN Table_B B
ON A.Key = B.Key

RIGHT JOIN



This query will return all of the records in the right table (table B) regardless if any of those records have a match in the left table (table A). It will also return any matching records from the left table. This Join is written as follows:

SELECT <select_list>
FROM Table_A A
RIGHT JOIN Table_B B
ON A.Key = B.Key

In the model_beekeeper, you find an example of JOIN. In this case there should be a match between beekeeper and and document, unless the line will not be qualified. But for apiary table, there should be a match or not. If there is no apiary in relation, the line will still be qualified.

```
public function getBeekeeperDocument($id)
{
        $sql = "SELECT document.*,name,directory,location
        FROM document
            LEFT JOIN document_type ON document.document_type_id = document_type.id
            LEFT JOIN apiary ON document.apiary_id = apiary.id
            LEFT JOIN colony ON document.colony_id = colony.id
            JOIN beekeeper ON document.beekeeper_id = beekeeper.id
        WHERE document.beekeeper_id = ?";
        $query = $this->db->query($sql, array($id));
        return $query->result_array();

}
```

Comparison operator

Name	Description
BETWEEN ... AND ...	Check whether a value is within a range of values
COALESCE()	Return the first non-NULL argument
=	Equal operator
<=>	NULL-safe equal to operator
>	Greater than operator
>=	Greater than or equal operator
GREATEST()	Return the largest argument
IN()	Check whether a value is within a set of values
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
<	Less than operator
<=	Less than or equal operator
LIKE	Simple pattern matching
NOT BETWEEN ... AND ...	Check whether a value is not within a range of values
!=, <>	Not equal operator
NOT IN()	Check whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

Database and CodeIgniter

CodeIgniter comes with a full-featured and very fast abstracted database class that supports both traditional structures and Query Builder patterns. The database functions offer clear, simple syntax.

Database configuration

The database configuration is in **application/config/database.php**. You need to indicate where is the server of the database, the username, password and name of the database.

```
$db['default'] = array(
    'dsn'   => '',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => '',
    'database' => 'meb',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
```

Connecting the database

In the system, the database is automatically connected when we start the system, in **application/config/autoload.php**

```
$autoload['libraries'] = array('session', 'form_validation', 'database', 'zip', 'calendar', 'table', 'Pdf');
```

Query

$this->db->query()

To submit a query, we use the query function. ($this_db->query). In this example, a parameter is passed to the query to get one row or all the rows. In the case we expect one row, we will use **row_array()** for the return. For many rows, it will be **result_array()**

```php
public function getNationalityData($id = null)
{
    if($id) {
        $sql = "SELECT * FROM nationality WHERE id = ?";
        $query = $this->db->query($sql, array($id));
        return $query->row_array();
    }

    $sql = "SELECT * FROM nationality";
    $query = $this->db->query($sql);
    return $query->result_array();
}
```

Inserting data

$this->db->insert()

Generates an insert string based on the data you supply, and runs the query. You can either pass an array or an object to the function.

```php
public function create($data)
{
    if($data) {
        $insert = $this->db->insert('nationality', $data);
        return ($insert == true) ? true : false;
    }
}
```

In application/controllers/Nationality.php an array is created to pass $data.

```php
$data = array(
    'name' => $this->input->post('nationality_name'),
    'active' => $this->input->post('active'),
);

$create = $this->model_nationality->create($data);
```

Updating data

$this->db->update()

Generates an update string and runs the query based on the data you supply. You can pass an array or an object to the function. Here is an example using an array:

```php
public function update($data, $id)
{
    if($data && $id) {
        $this->db->where('id', $id);
        $update = $this->db->update('nationality', $data);
        return ($update == true) ? true : false;
    }
}
```

In this case we pass the $data, which is an array created in application/controllers/Nationality.php and also the $id of the row of table Nationality we want to update.

```php
if ($this->form_validation->run() == TRUE) {
    $data = array(
        'name' => $this->input->post('edit_nationality_name'),
        'active' => $this->input->post('edit_active'),
    );

    $update = $this->model_nationality->update($data, $id);

    if($update == true) {
        $response['success'] = true;
        $response['messages'] = $this->lang->line('Successfully updated');
    }
    else {
        $response['success'] = false;
        $response['messages'] = $this->lang->line('Error in the database while updating the information');

    }
```

Deleting data

$this->db->delete()

Generates a delete SQL string and runs the query.

```php
public function remove($id)
{
    if($id) {
        $this->db->where('id', $id);
        $delete = $this->db->delete('nationality');
        return ($delete == true) ? true : false;
    }
}
```

```php
//---> Validate if the nationality is used in table Beekeeper
public function checkIntegrity($id)
{
    // select with the wildcard %.  It is possible to have more
    // than one nationality in beekeeper table.   In this case, the information
    // will appear between bracket ["1"].  The search will be
    // SELECT * FROM beekeeper WHERE nationality_id LIKE '%["1"]%'
    $this->db->select('*');
    $this->db->from('beekeeper');
    $this->db->like('nationality_id', $id, 'both');
    $query = $this->db->get();
    return $query->num_rows();

}
```

In the controller, we check the integrity of the database before deleting a row.

```php
//---> Validate if the information is used in beekeeper table
$total_beekeeper = $this->model_nationality->checkIntegrity($nationality_id);
//---> If no beekeeper have this information, we can delete
if ($total_beekeeper == 0) {
    $delete = $this->model_nationality->remove($nationality_id);
    if($delete == true) {
        $response['success'] = true;
        $response['messages'] = $this->lang->line('Successfully deleted');}
    else {
        $response['success'] = false;
        $response['messages'] = $this->lang->line('Error in the database while deleting the information');}
    }
```