```
                $this->dogs_array["dog"][$dogs_array_size + $I] = $records_array[$I];
                $dog_id = rand(0,9999); // get a number between 0 and 9999
                while (in_array($dog_id, $this->dogs_array, true)) // in array?
                { $dog_id = rand(0,9999); // if it is get another number
}
$chge_string .="INSERT INTO Dogs VALUES('";
                $chge_string .= $dog_id . "', '" . $records_array[$I]['dog_name'] . "', '";
                $chge_string .= $records_array[$I]['dog_weight'] . "', '";
                $chge_string .= $records_array[$I]['dog_color'] . "', '";
                $chge_string .= $records_array[$I]['dog_breed'] . "');";
        }
        $chge_log_file = date('mdYhis') . $this->change_log_file;
        error_log($chge_string,3,$chge_log_file); // might exceed 120 chars
}
```

If you review the `changeRecords` method, a SQL `WHERE` clause was built using a property named `dog_id`. In the XML and JSON examples you did not have this field. However, SQL `UPDATE` requires a `where` clause to determine which record(s) to update. The property used needs to be unique to identify the exact record(s). The only place the code must generate this `dog_id` is when a new record is created in the database (in the `insertRecords` method). This can be done using the PHP `rand` method.

The PHP `rand` method produces random numbers. The first parameter is the starting number (0) and the second parameter is the last number (9999). The size of this field is set to `char(4)` in the database, which allows up to four characters. This would allow you up to 10,000 dogs. I am sure that will be more than enough!

The `while` loop in the `insertRecords` method uses the PHP `in_array` method to determine if the number is already in the `dogs_array` (which contains all the current records in the database). A third parameter, which determines if a `strict` **search** (comparing data types) should occur, must be set to produce reliable results with multidimensional associate arrays. If the number does exist, the logic continues to generate a new random number until a unique one is found. The value is then placed in `$dog_id`, which will be inserted into the database along with the other fields (`dog_name`, `dog_weight`, `dog_color`, and `dog_breed`). Note: This code assumes that the `Dogs` table in the database has been created with the fields in the order shown (`dog_id`, `dog_name`, `dog_weight`, `dog_color`, and `dog_breed`).

The change log (which is now also a SQL script file) would now contain statements similar to the following:

```
INSERT INTO Dogs VALUES('2288', 'tester1', '19', 'Green', 'Lab');
UPDATE Dogs SET dog_name='tester1', dog_weight='19', dog_color='Green',
dog_breed='Lab' WHERE dog_id='0111';
UPDATE Dogs SET dog_name='tester2', dog_weight='19', dog_color='Green',
dog_breed='Lab' WHERE dog_id='1211';
DELETE FROM Dogs WHERE dog_id='1111';
```

This file can be run against the database when all changes have been logged. The destructor can now execute this file (instead of removing the table and inserting all the records back into a new table).

```
$mysqli = new mysqli($server, $db_username, $db_password, $database);
if ($mysqli->connect_errno)
{
          throw new Exception("MySQL connection error:" . $mysqli->connect_error);
}
```