CVWO Assignment 2

Shen Yichen

The aim of this assignment is to expand the HelloWorld module in the CVWO sources to allow for mass import of contacts, which comes in a name, email pair.

Set-up

The database is set up in MySQL with a additional user account.

```
User: drupal
Password: drupal
Database: drupal
```

Issues

There was an issue when enabling the CVWO core module. When creating the table `cvwo_address`, there was an error from PDO:

```
PDOException: SQLSTATE[42000]: Syntax error or access violation
   : 1067 Invalid default value for ‘date_create’
```

The issue seem to stem from how `CURRENT_TIMESTAMP` is handled by Drupal and MySQL. When two `TIMESTAMP` columns are defined as `NOT NULL` without any default values, the first column's time-stamp is set to refresh automatically when the record is updated. The second column however, would be set to 0 as default. This behaviour was due to older versions of MySQL restricting the `CURRENT_TIMESTAMP` mechanism to a single column.

A workaround is used. One of the columns is first removed from the schema in CVWO core. The module is enabled, and then the column is added back manually. Since current versions of MySQL has lifted the restriction for `CURRENT_TIMESTAMP` to be defined on a single column only, both columns are set to use `CURRENT_TIMESTAMP`.

Use Case

The HelloWorld module would be expanded with a single feature. The user would be able to see a new tab named "Mass Import".

"Mass Import" contains a text area which accepts CSV values for {name, email} pairs. The PHP function `str_getcsv` will be used to parse the values, as it will automatically handle escape values.

Merging

There are cases where in a mass import, repeated values are present. There are a few ways to handle this issue.

Firstly, repeated values would simply overwrite the existing values. A warning would be posted on the UI form to warn users of this behaviour. The implementation here is the simplest, as described below. Such an implementation though, will not raise any warnings if records are erroneously replaced. The user though, would be able to correct such errors easily by editing the input only at the error line. He could then simply reimport everything without error.

The second way would be to reject repeated values. This method has the advantage of not accidentally overriding values. However, it brings complications, namely, what to show the user when a record fails to be inserted. As described below, showing more information about the failures might also introduce significant overhead to the system. Furthermore, this method would provide no easy way for mass updating of the records; each old record must be manually removed one by one before the new records are imported, and thus may actually be less user friendly.

If time permits, I may look into allowing for both options in the form.

Implementations

Database

The implementation here corresponds to the above merging description.

To implement the first case, we simply use the merge wrapper supplied in the CVWO base database API 2. A new method in the HelloWorld API would need to be defined though. In the new method, the merge object returned will be supplied with all of the data to be imported at once, using the `values()` method. This configuration would structure the data to be merged via a single query, saving the overhead of multiple query requests to the database.

The second case is somewhat trickier. If we were to keep the idea of aggregating the data into a single query, we could still follow the above outline, but with the insert wrapper this time. In that case, we will also pass `array('return' => DATABASE::RETURN_AFFECTED)` as an option to the insert wrapper. Doing so allows us to get the number of rows inserted from the `execute()` function, thus informing the user of the number of rows which failed to insert.

We could also simply place the `add_user` function from the HelloWorld API in a loop and add each user one by one. Doing so allows us to pinpoint and display which users could not be inserted, and subsequently allows them to be displayed to the user. However, doing so will cause numerous queries to the database distinctly, and introduces significant overhead.

The first method would be my priority when attempting this assignment.

UI

The UI will be done with a Drupal form much in the same way the add user page was done. As mentioned, it will contain a single text area for the mass import record, in addition to a input box for optional comments. The hooks will be defined in a separate `.inc` file and the tab specified in the module file.

The validation will check for the format of the emails as usual, and would reject the entire input if any record is found to be invalid. CSV conversion will thus happen in the validation too, to obtain the email address. As such, any invalid CSV formats will also be caught and an error thrown.

Submission would insert the values into the database with one of the methods as specified above. Warnings will be thrown depending on the method adopted. For example, merge would unlikely generate any warnings. Insertion would remind either of how many records are inserted or which records are not inserted due to key duplication.