JSON Data

JSON data would require some changes to accommodate the additional fields.

```
{"user":[
{"userid":"Fredfred","password":"$2y$10$VosI32FejL.bOMaCjGbBp.Jre6Ipa.tLYQrVqj9kiVpef5zZ25qQK",
"datestamp":"2015-09-03","attempts":"0","lastattempt":"08052015044229","validattempt":
"08052015045431"},
{"userid":"Poppoppop","password":"$2y$10$C1jXhTlOmyamuLKhZxK5m.4X4TVcdeFbeLSBIA7l4fx6tUnC8vrg6",
"datestamp":"2015-09-04","attempts":"2","lastattempt":"08062015011347","validattempt":
"08062015113038"}
]}
```

The `$newupstring` would also require changes due to these new fields. The location of some of the code would also move to methods, as mentioned earlier.

```
$newupstring = ',{"userid":"' . $user['userid'] . '","password":"' . $user['password'] . '","';
$newupstring .= 'datestamp":"' . $user['datestamp'] . '","attempts":"' . $user['attempts'] . '","';
$newupstring .= 'lastattempt":"' . $user['lastattempt'] . '","validattempt":"' .
$user['validattempt'] .'"';
 $newupstring .= '"}\n]}';
```

MySQL Data

MySQL login code requires the `UPDATE` statement (instead of an `INSERT` statement, as seen in the registration code) to update any fields that have changed. In addition, the `clean_input` method from Chapter 4 should to be used to remove any harmful PHP and SQL statements from the `userid` field. This will help reduce the chance of *SQL injection* occuring, which could cause the SQL statement to change more than just the required fields and record(s). For example, if the `$user['userid']` field contained `'*'`, all records would be updated instead of just one record with a valid user ID.

The user ID could also be validated as existing in the database before executing the `UPDATE` statement. This (assuming all data in the database is valid) would also reduce the chance of harmful changes.

In the following example, all possible files are updated at once. Alternatively, only those fields that change could be updated when needed. However this would require more code and not necessarily be any more efficient. Also, if the user ID is not validated as existing in the database beforehand, the SQL statement will automatically not update the fields if `userid` is not in the database.

```
$userid = clean_input($user['userid']);
$sql ="UPDATE Users SET(datestamp='" . $user['datestamp'] . "',attempts='";
$sql .=$user['attempts'] . "',lastattempt='" . $user['lastattempt'] . "',";
$sql .="validattempt='" . $user['validattempt'] . "') WHERE userid='" . $userid . "';";
```

Notice that the `UPDATE` code does not include the password field in the `WHERE` statement. In this example, some fields would be updated when the password is not valid, and some would be updated when the password is valid. If the SQL statement is broken into multiple statements (at least one for valid user ID/password and one for not valid user/password), then the `WHERE` statement for the valid information can include both the user ID and password, and the statement for non-valid information can just include the password. An example of a complete login, registration, and password-change application using a MySQL database is included on the book's web site under Chapter 7.