**Sparrowhawk – Technical Manual**

**List of files in distribution:**

Bootstrap themes for page styling

New report page

New owner

External search script

External search script

People search page

(ignore)

Mysql/php connection script

Mysql/php connection script

(ignore)

(Edit report page)

Add fines page

Home page

Home page mascot

Index (initiates login)

Logout script

(ignore)

New user registration page

Report menu page

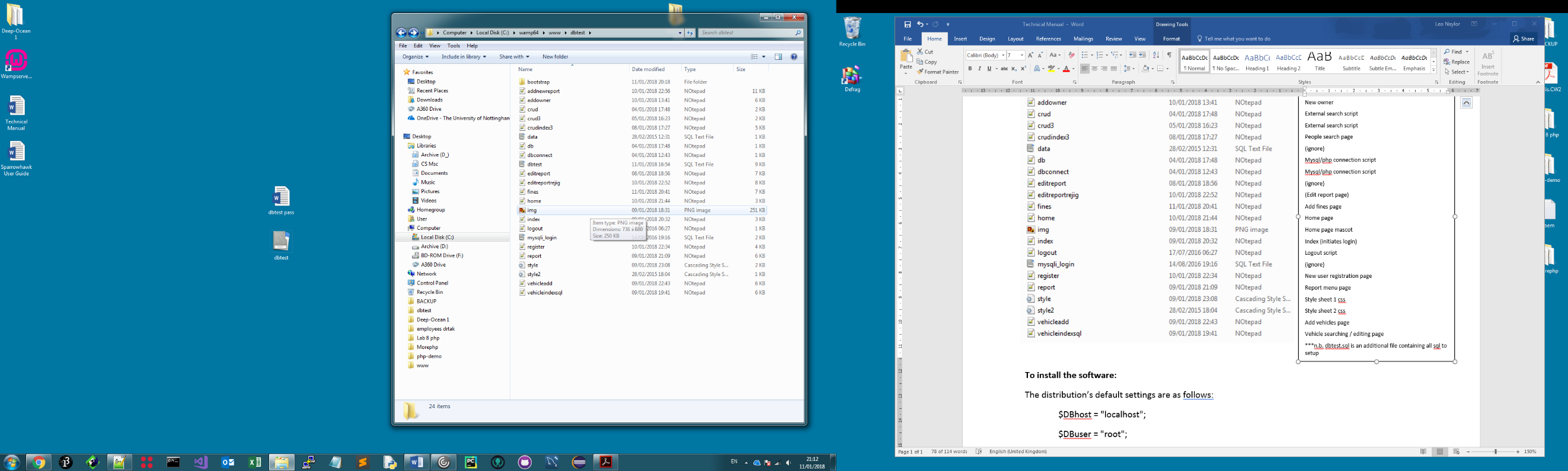
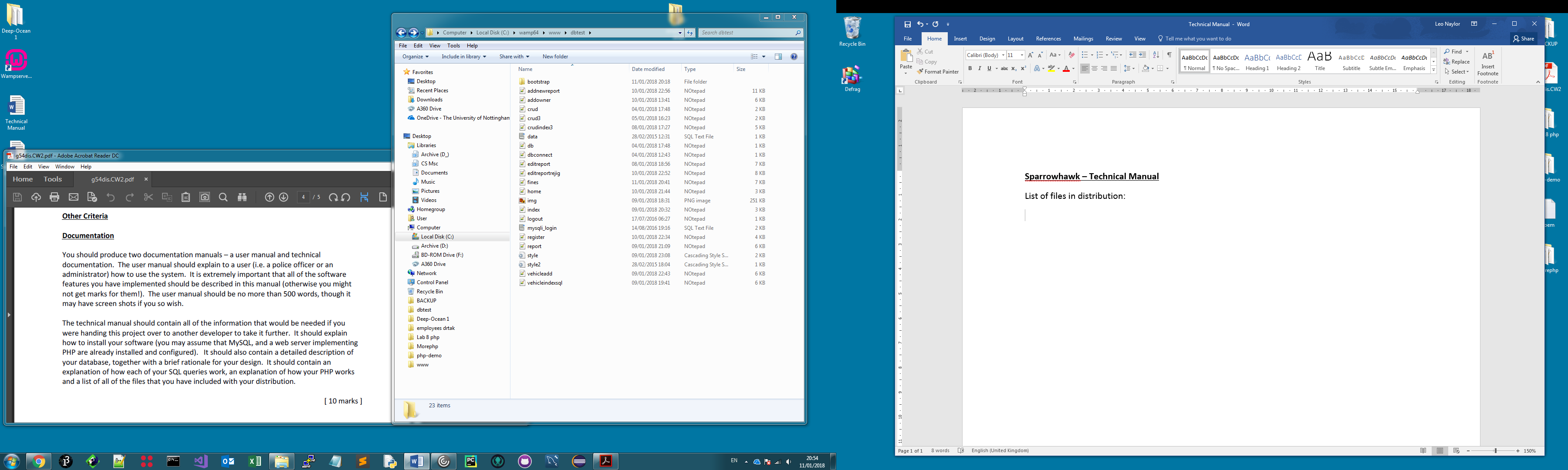
Style sheet 1 css

Style sheet 2 css

Add vehicles page

Vehicle searching / editing page

\*\*\*n.b. dbtest.sql is an additional file containing all sql to setup the database



**To install the software:**

The distribution’s default server login settings are as follows:

$DBhost = "localhost";

$DBuser = "root";

$DBpass = "";

$DBname = "dbtest";

These settings must be used for the database, otherwise the distribution connection scripts and pages queries will need to be updated accordingly to match alternative server login settings.

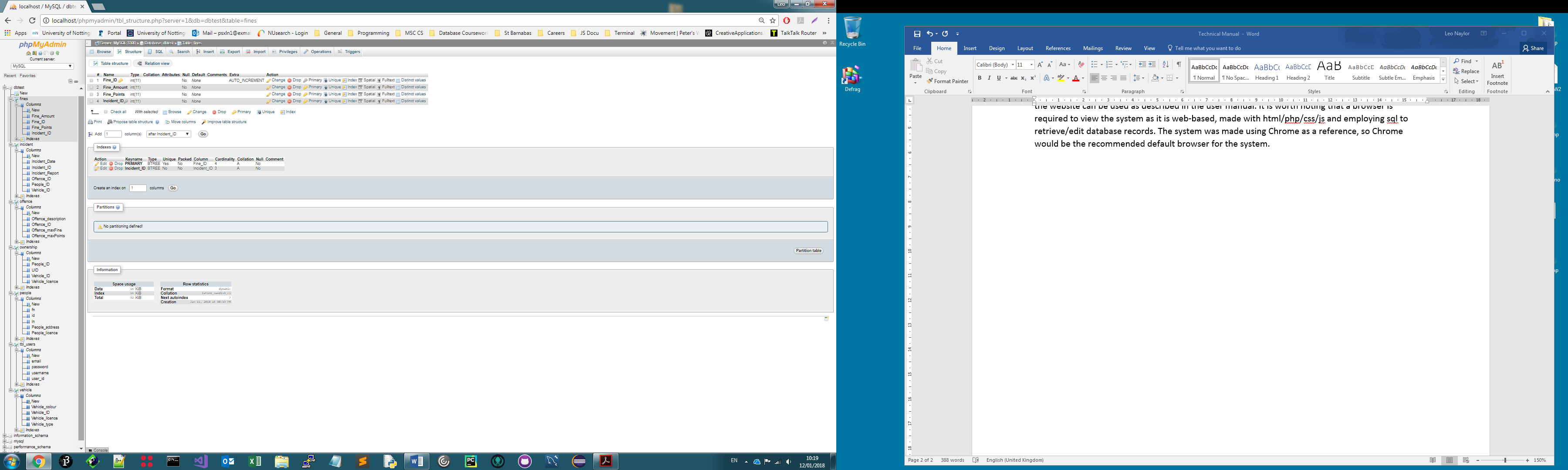
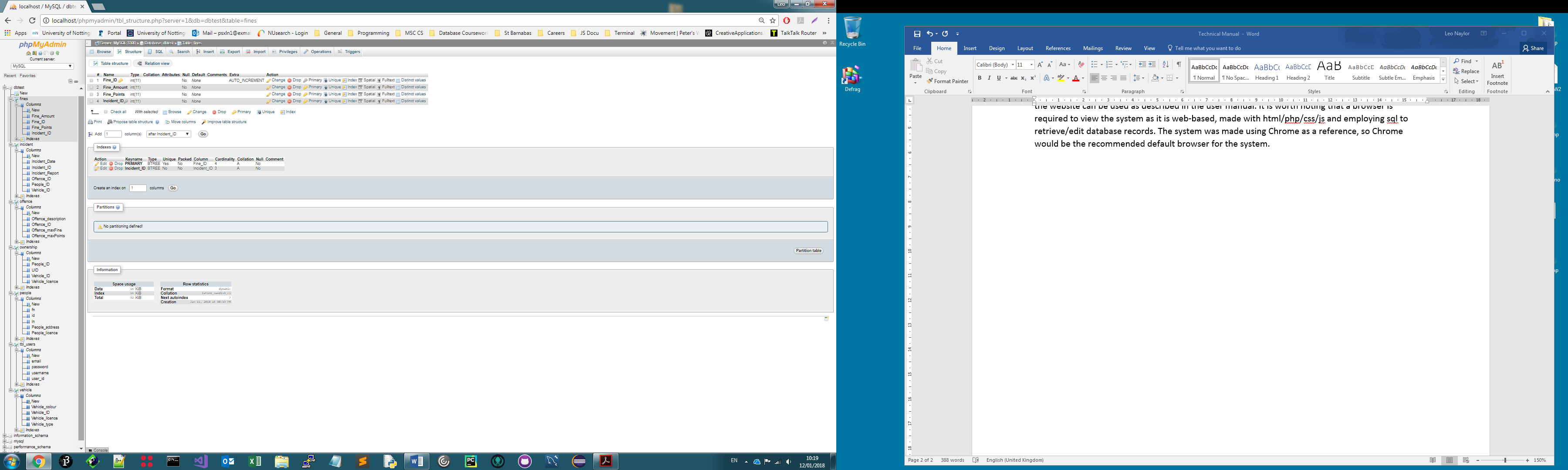
The user must setup a database called ‘dbtest’ using the dbtest.sql file provided in the distribution, this can be done easily via software like phpmyadmin or mysql workbench by creating the new database and using the built-in import functions. After clicking ‘import’ the user can select dbtest.sql from the file browser and all tables will be created in the ‘dbtest’ database.

When this has been done, the distribution folder can be copied into the public html folder; in WAMP this is ‘www’, in XAMP this is ‘htdocs’, on the Nottingham University systems this is public\_html but all people are discouraged from hosting their files on the student public\_html folder as it cannot be accessed externally and changes the way pages are displayed!

Once the folder has been created in the server directory the Sparrowhawk system can be accessed by entering <http://localhost/dbtest/index.php> in the browser, with most local server software merely typing the folder’s directory is enough to initiate the site’s movement to the index.php page. If using a Nottingham University machine <http://avon.cs.nott.ac.uk/~psxln1/dbtest/index.php> or similar -depending on your account name- would be the appropriate address to use. From this point the website can be used as described in the user manual. It is worth noting that a browser is required to view the system as it is web-based, made with html/php/css/js and employing sql to retrieve/edit database records. The system was made using Chrome as a reference, so Chrome would be the recommended default browser for the system.

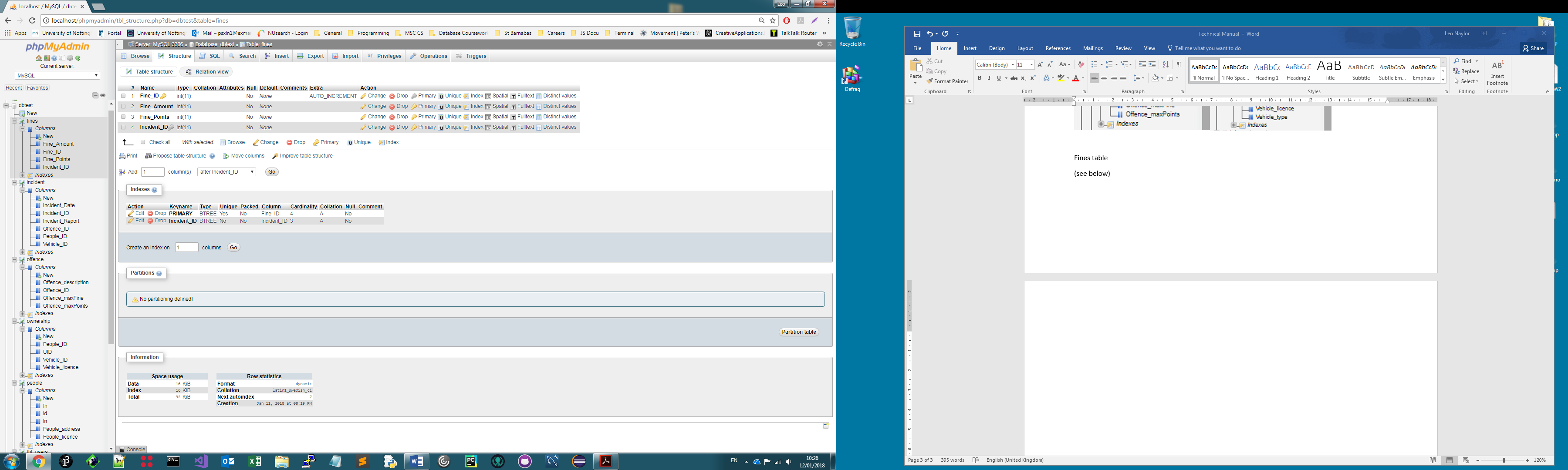
**Database commentary**

**-Structure**



**Fines table**

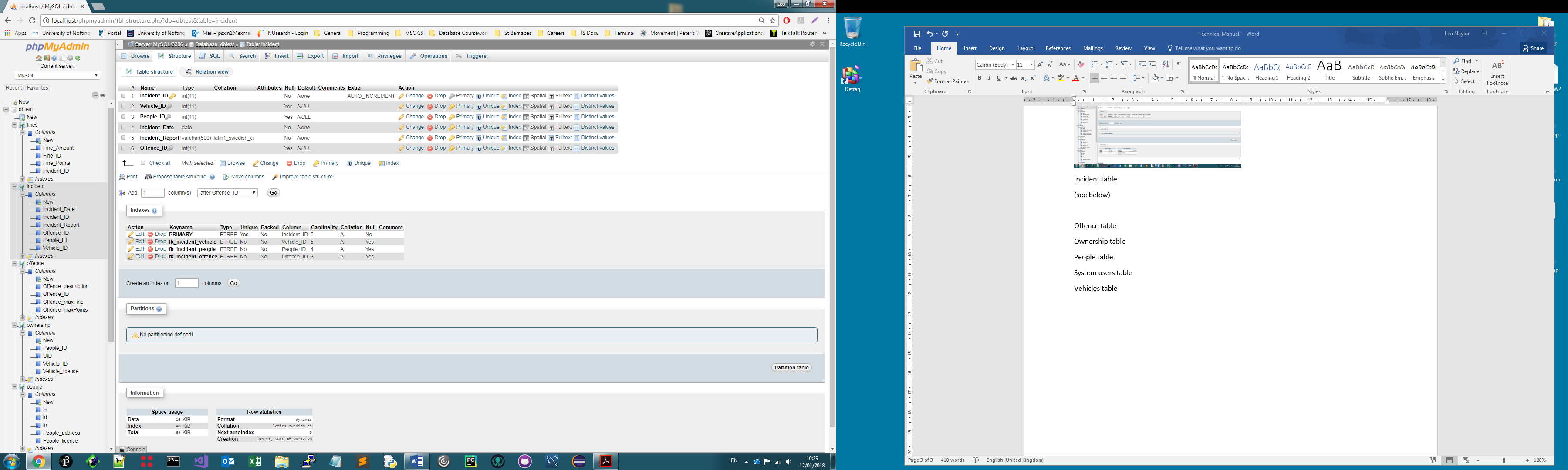
**(see below)**



Fine\_ID is set to auto increment and provides a primary key for the table, Incident\_ID has a foreign-key (del: restrict, update: restrict) constraint with incident.Incident\_ID, all fields are int type.

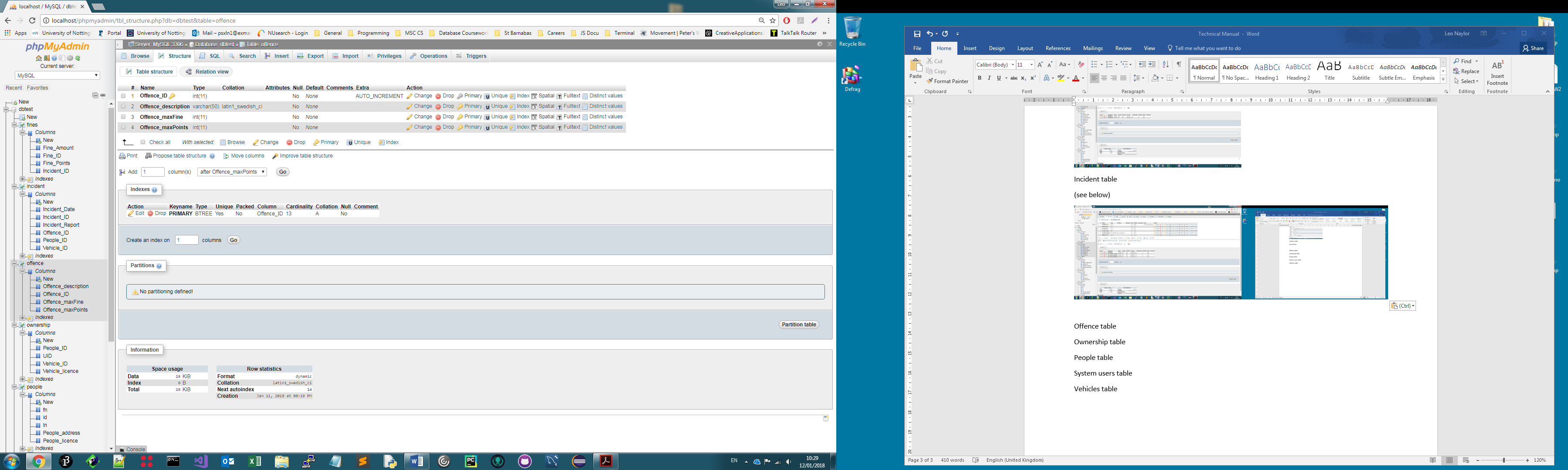
**Incident table**

**(see below)**



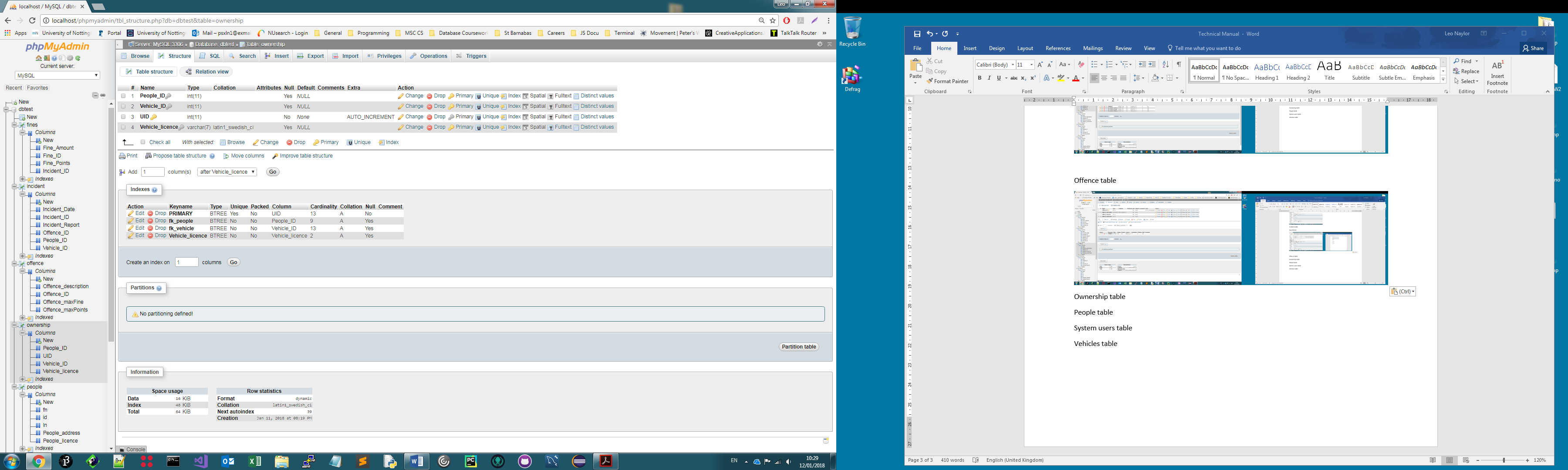
Incident\_ID is seta primary key with auto-increment, Offence\_ID has foreign key offence.Offence\_ID, column types have been altered accordingly. Some null values have been permitted to allow sequential SQL statements to be performed via the PHP, but redundancy is avoided as the user interface and PHP actions will not allow null values to be lodged in the table.

**Offence table**



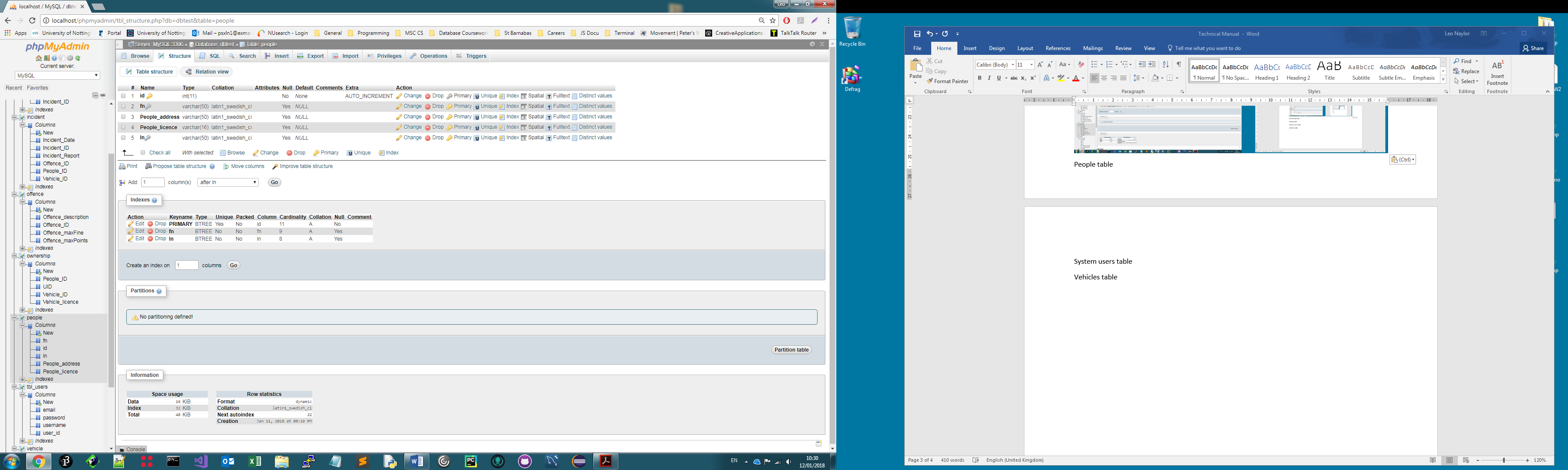
Offering a fairly simple structure, the offence table acts as a dictionary of offences, points and fines. The offence\_id is the primary key set to auto increment. This table has no parent/child relations.

**Ownership table**



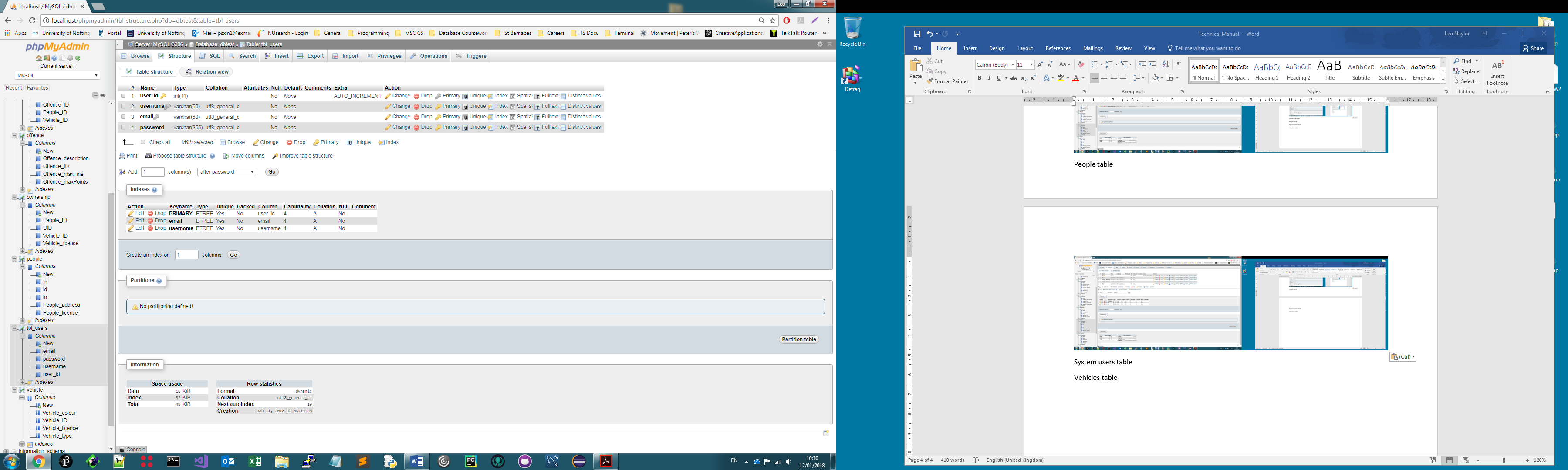
People and vehicles IDs have foreign keys with their respective parent tables vehicles and people. UID provides an auto-incremented primary key. Vehicle\_licence is not used actively as SQL refers to the vehicles table for this. The ownership table essentially combines elements of the people and vehicles tables in order to prevent many to many relationships. It is updated any time a vehicle is added to the system. Vehicles are not allowed to be added to the system unless they are assigned an owner, this functionality is somewhat restrictive, but as in reality all vehicles must be licensed to an owner, taxed, insured, or declared SORN (even when SORN they are still registered to an owner), it does make sense to operate the system in this way and it eliminates dangles, nulls, redundancies etc. Vehicle\_licence is limited to 7 characters.

**People table**



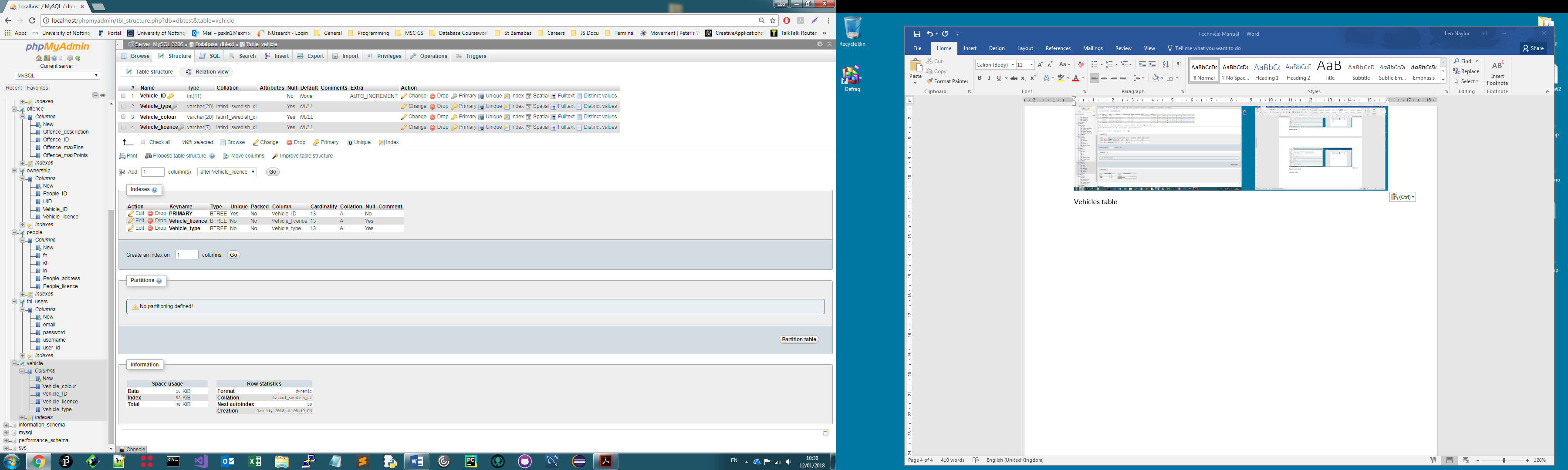
The people\_ID table does not have any external relationships, id provides primary key and a.i., types and field lengths have been adjusted accordingly. Null values have been prevented on the front end, so no restrictions have been necessary here.

**System users table**



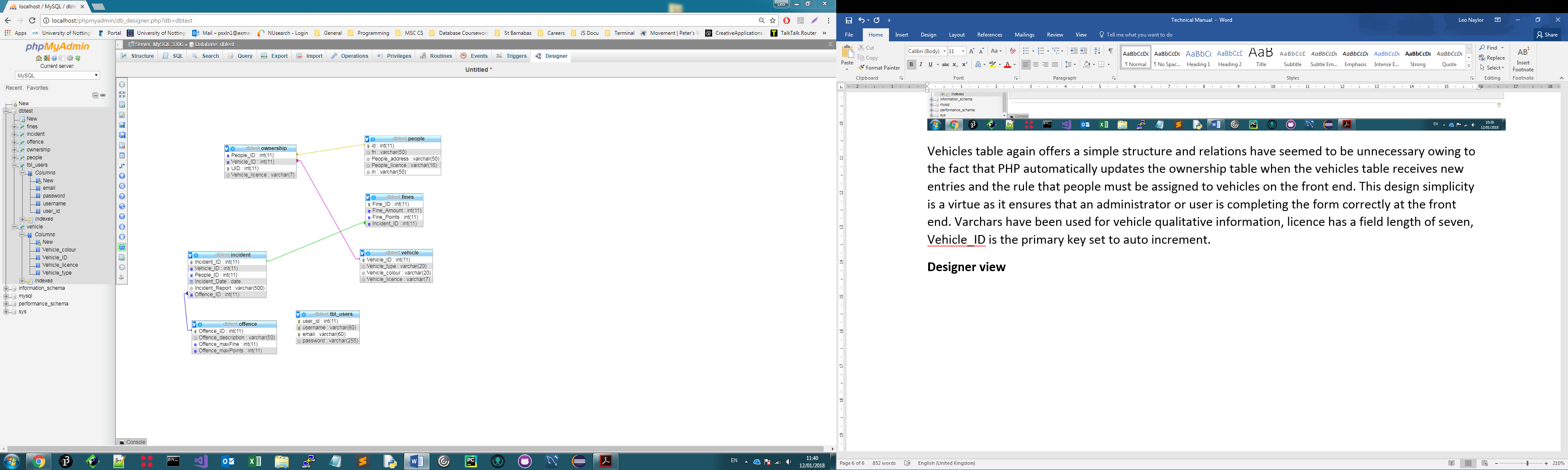
User\_id is set as primary key auto increment, this table is used by the login screen to verify user account details and drawn session variables for the user’s session, it also allows the system to distinguish whether it is in normal/admin mode (haskins is the only administrator currently allowed).

**Vehicles table**



Vehicles table again offers a simple structure and relations have seemed to be unnecessary owing to the fact that PHP automatically updates the ownership table when the vehicles table receives new entries and the rule that people must be assigned to vehicles on the front end. This design simplicity is a virtue as it ensures that an administrator or user is completing the form correctly at the front end. Varchars have been used for vehicle qualitative information, licence has a field length of seven, Vehicle\_ID is the primary key set to auto increment.

**Designer view**



It should be noted that the InnoDB engine has been used in order to allow relations/foreign keys to be used.

**Design rationale**

In the nascent stages of the system design I thought about how the system would be navigated from the user’s perspective. It would appear that a modular site navigation bar is generally a very helpful visual cue and removes any uncertainty as to one’s whereabouts from a user perspective. This allowed me to come up with the concept of a growing navigation bar that introduces additional tiers for each level of operational complexity, i.e. 1. Option selection 2. A search query 3. An edit this leads to simple navigation, organised thinking, the user always knowing which aspect of the database they are interacting with.

A second design consideration was setting the user up for success. By this token any areas where the user is performing changes e.g. adding a report, they are given the opportunity to select existing vehicles and users based on license and registration. In the absence of these details each may be added via drop down menus so that the user must consciously circumvent the existing records if they wish to add details to the database, this encourages active thinking and removes the need for over-complication in terms of code and database queries, it also discourages redundancy and duplication. Whilst it would theoretically be possible for a user to enter spurious records, they would have to actively contravene what is a very clearly defined correct system use and this would be an abuse of the system. It would be possible to prevent a person from being entered into the people table twice by performing a check on licences using a formulation of PHP if…{} and SQL (WHERE EXISTS), but as this has not been asked for in the specification it has been left out of the final design, instead employing a system design that imposes the correct use practices on the user. Either way, the system still functions correctly based on the specification and presents the data effectively in accordance with the user guide.

**SQL queries / PHP code (n.b. images are high res so just zoom in (ctrl + mousewheel))**

N.B.

Some amendments were made to the error checking late on in the build in order to prevent repeat records of cars or people, this was done with the following statements across several pages.

Vehicles

if (isset($\_POST['Registration']) && $\_POST['Type']!="" && $\_POST['Colour']!="" && $\_POST['Owner']!="")

{

$car = $\_POST['Registration'];

$regcheck = mysqli\_real\_escape\_string($conn, $car);

$result2 = mysqli\_query($conn, "SELECT 1 FROM vehicle WHERE Vehicle\_licence='$car' LIMIT 1");

if (mysqli\_fetch\_row($result2)) {

echo 'Vehicle already in database';

}

People

if (isset($\_POST['Registration']) && $\_POST['Type']!="" && $\_POST['Colour']!="" && $\_POST['firstname']!="" && $\_POST['lastname']!="" && $\_POST['licence']!="" && $\_POST['addy']!="")

{

$own = $\_POST['licence'];

$dv = mysqli\_real\_escape\_string($conn, $own);

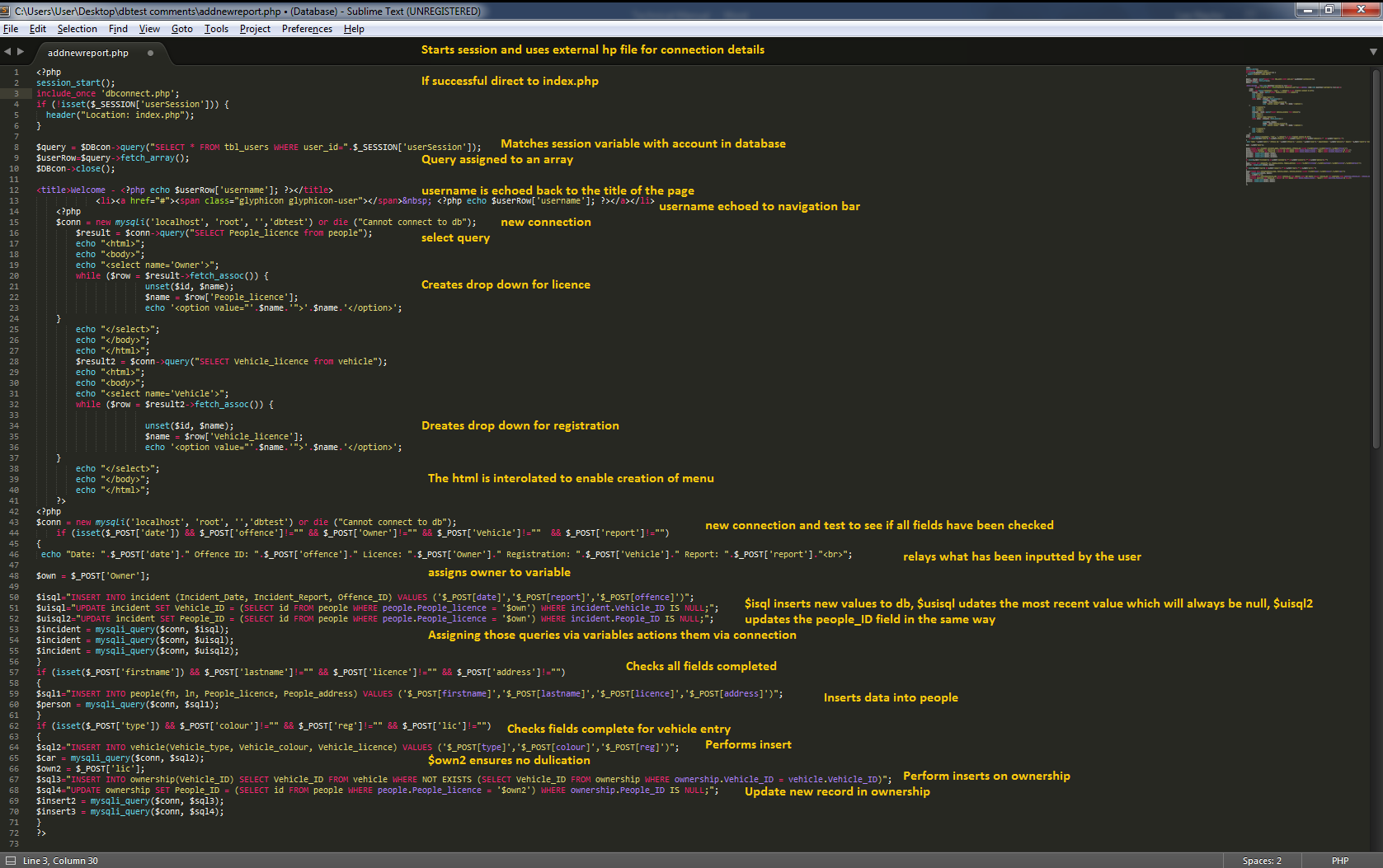
$result = mysqli\_query($conn, "SELECT 1 FROM people WHERE People\_licence='$own' LIMIT 1");

if (mysqli\_fetch\_row($result)) {

echo 'Person already in database';

As is ostensible, both methods refer to a candidate key which is guaranteed to be unique and enable the prevention of duplication through this means. This method is a straightforward way of checking whether a value has been returned via the php/sql comparison and outputting feedback for the user’s discretion.

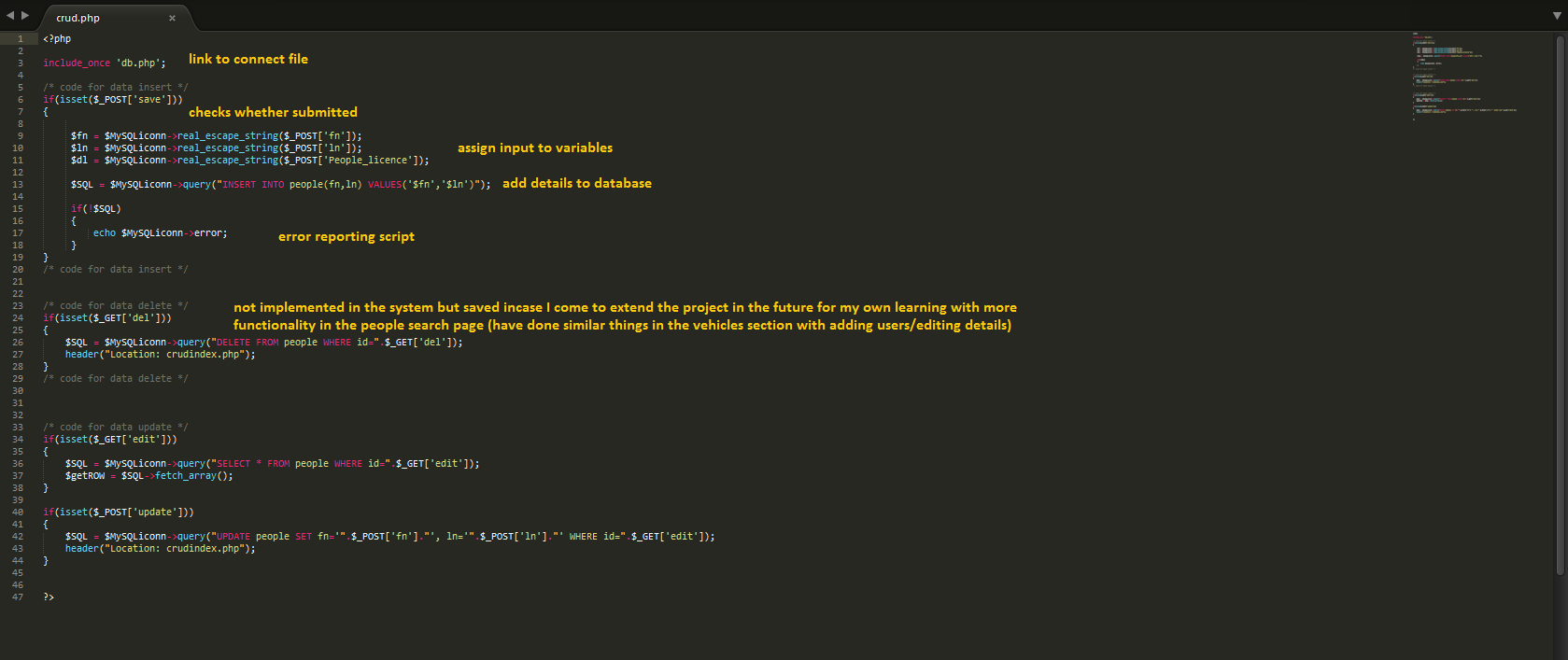
Addnewreport.php



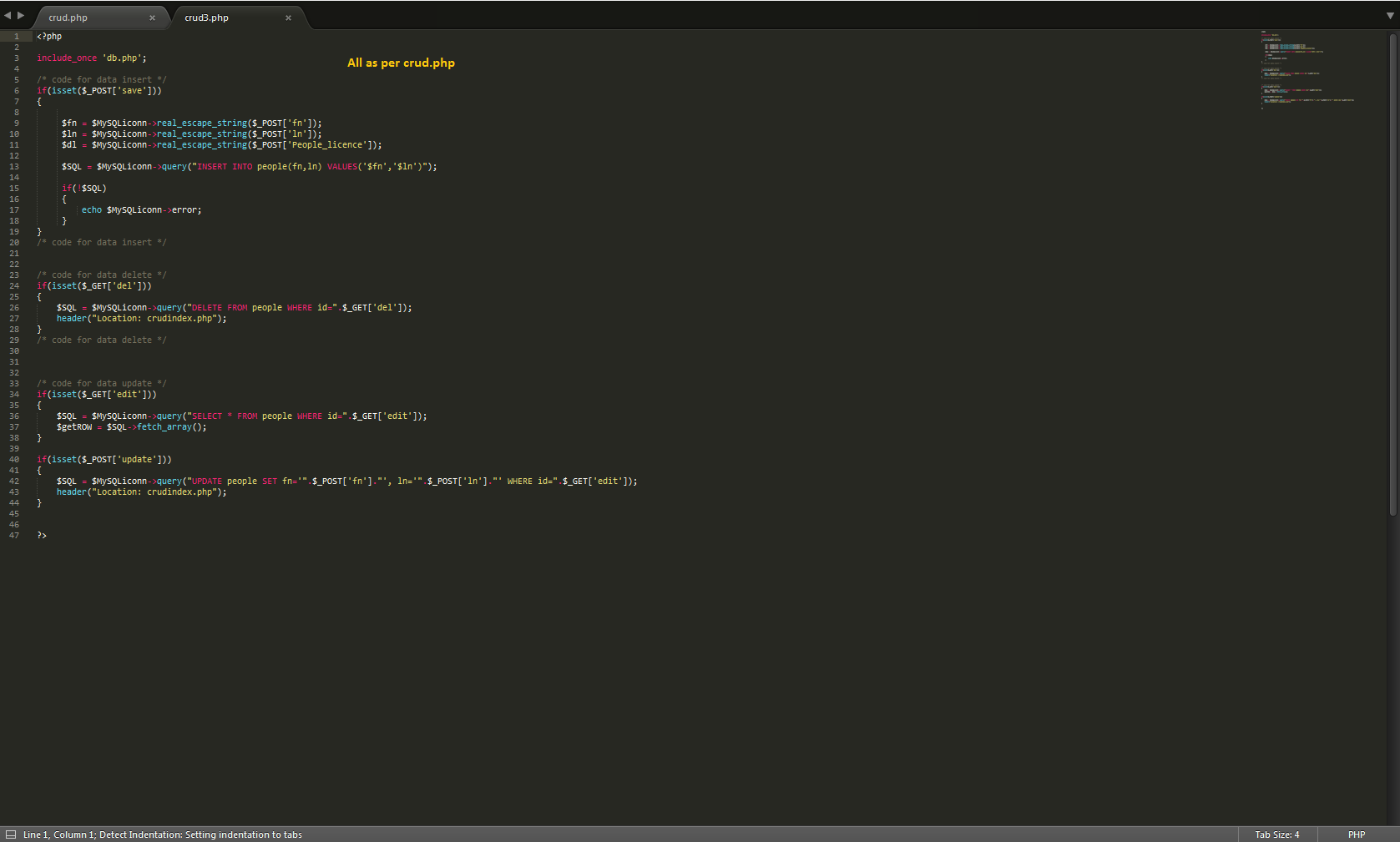
Addowner.php



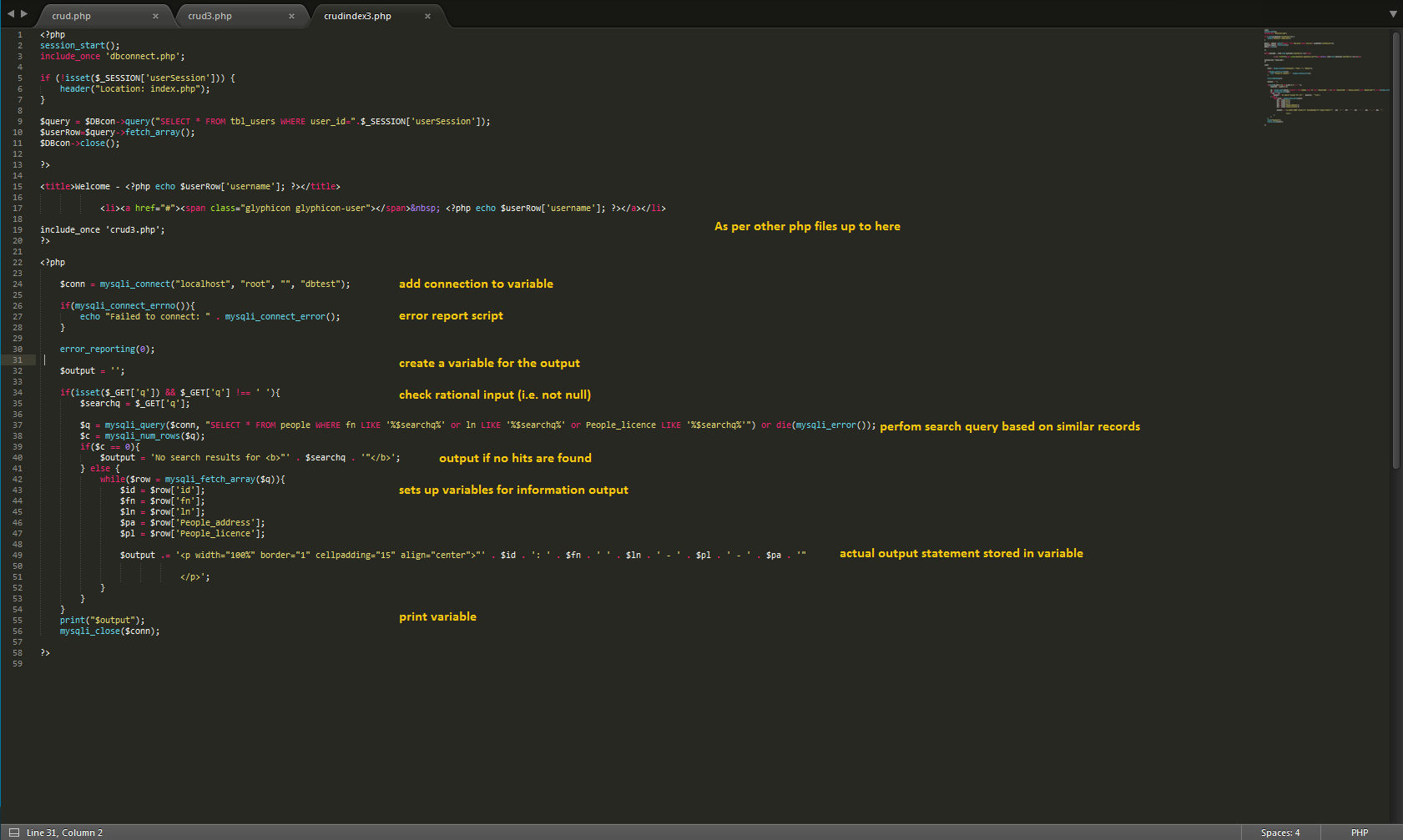
Crud.php



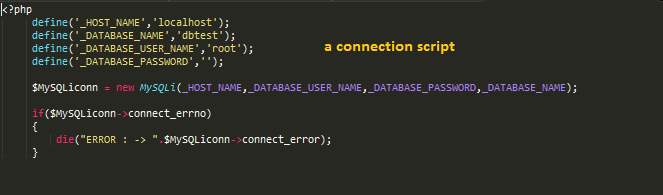
Crud3.php



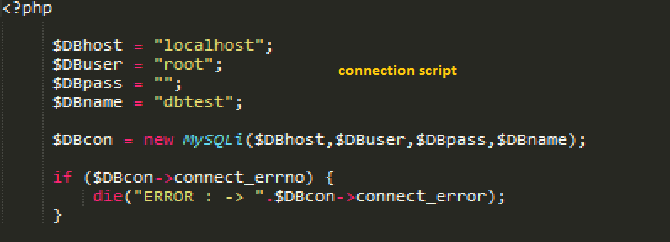
Crudindex3.php



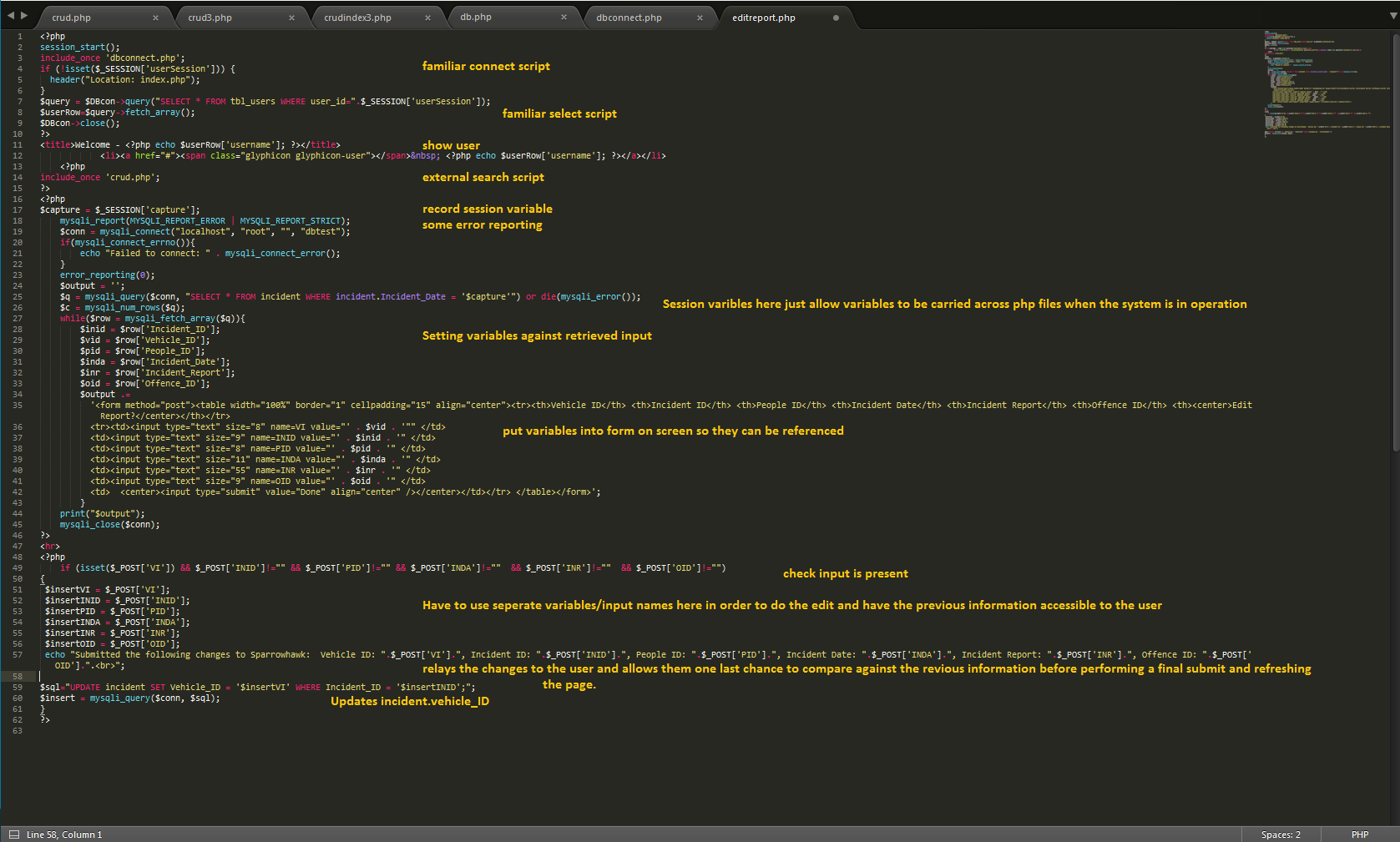
Db.php



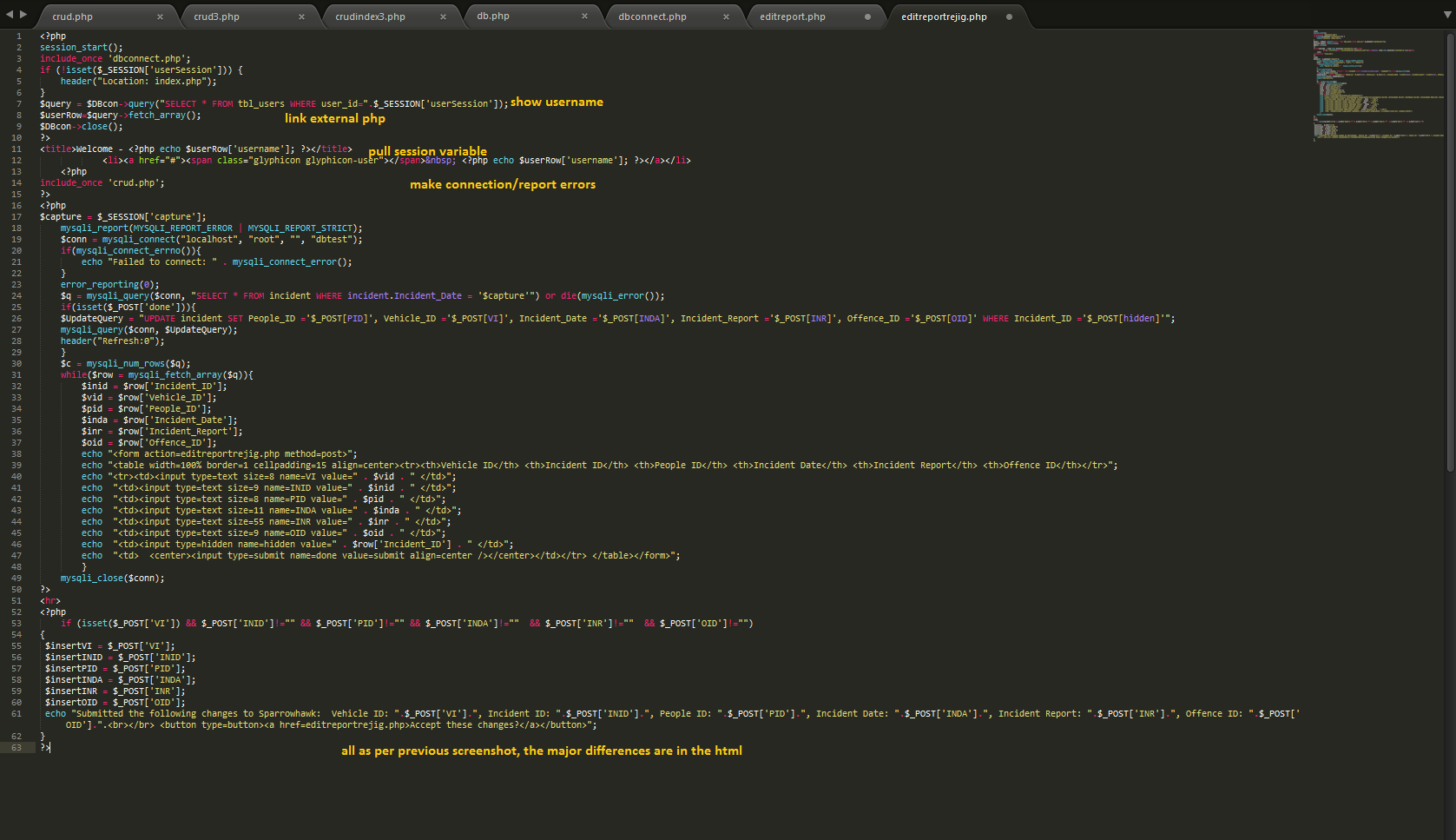
Dbconnect.php



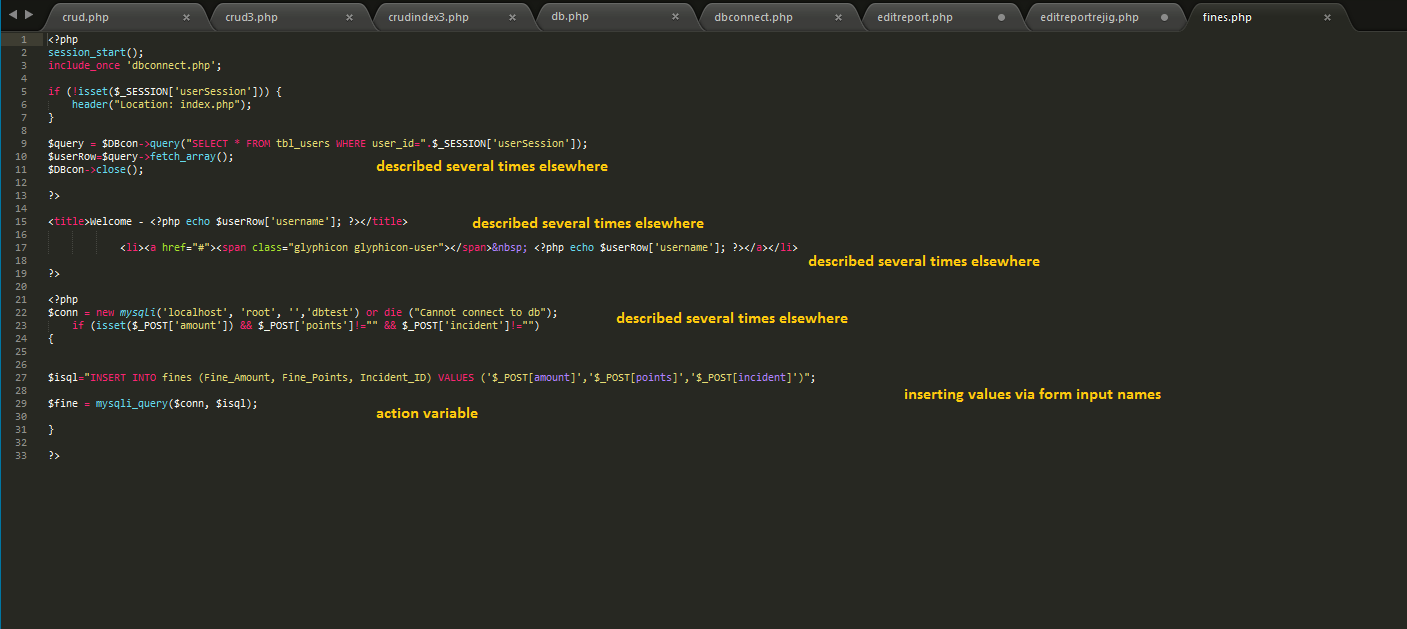
Editreport.php



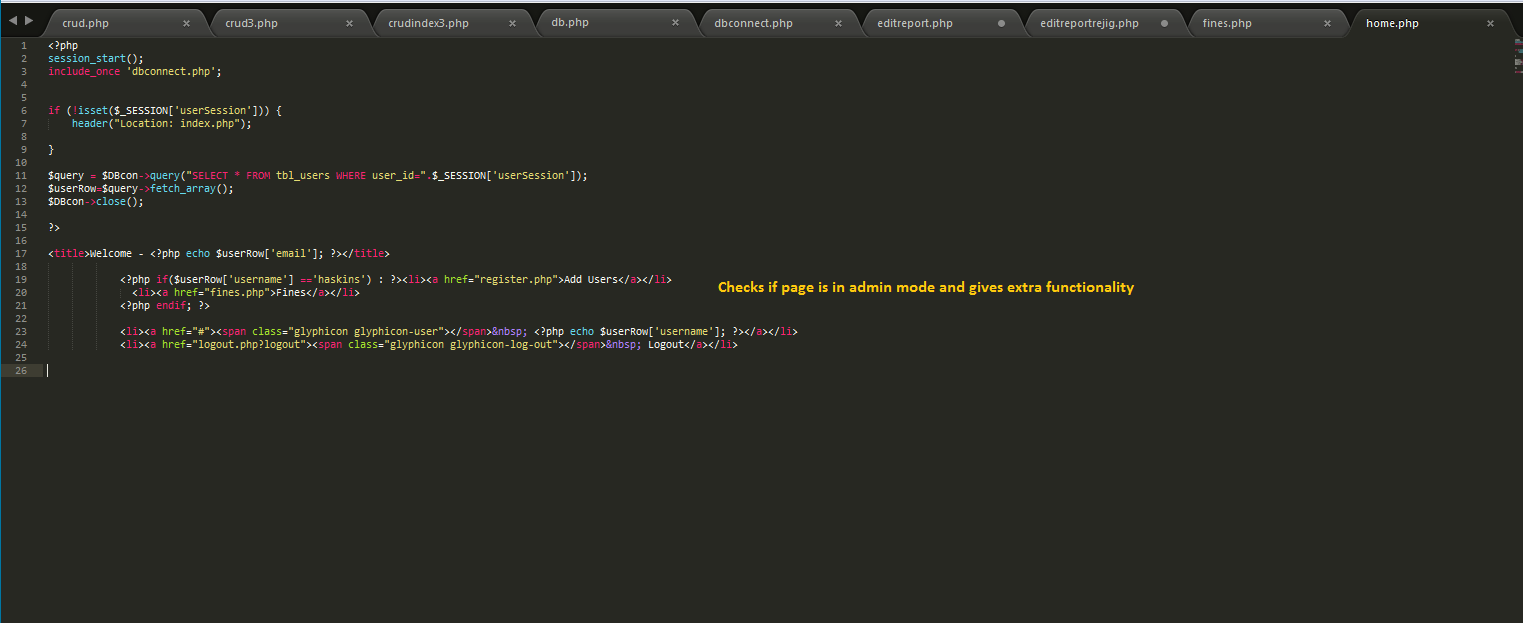
Editreportrejig.php



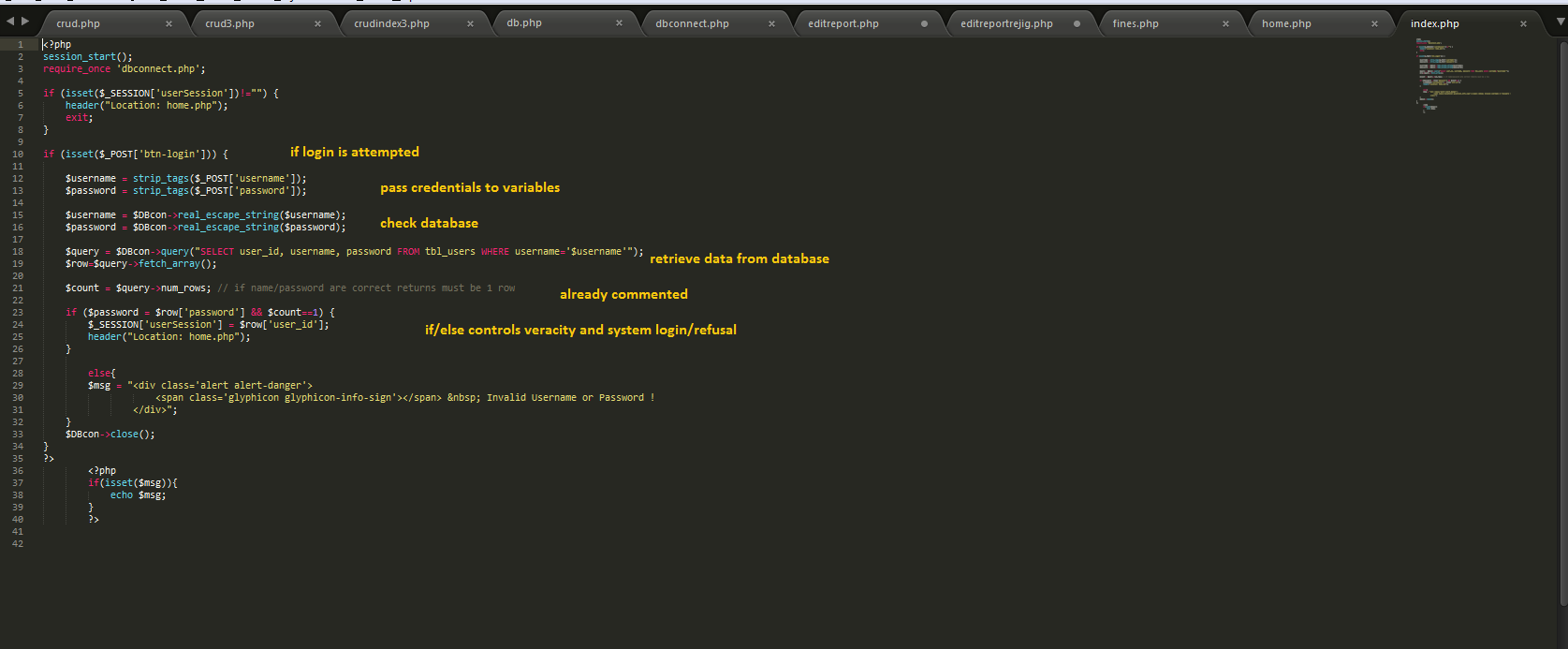
Fines.php



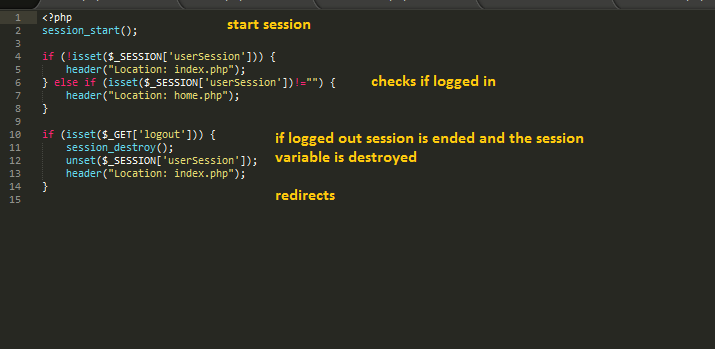
Home.php



Index.php



Logout.php



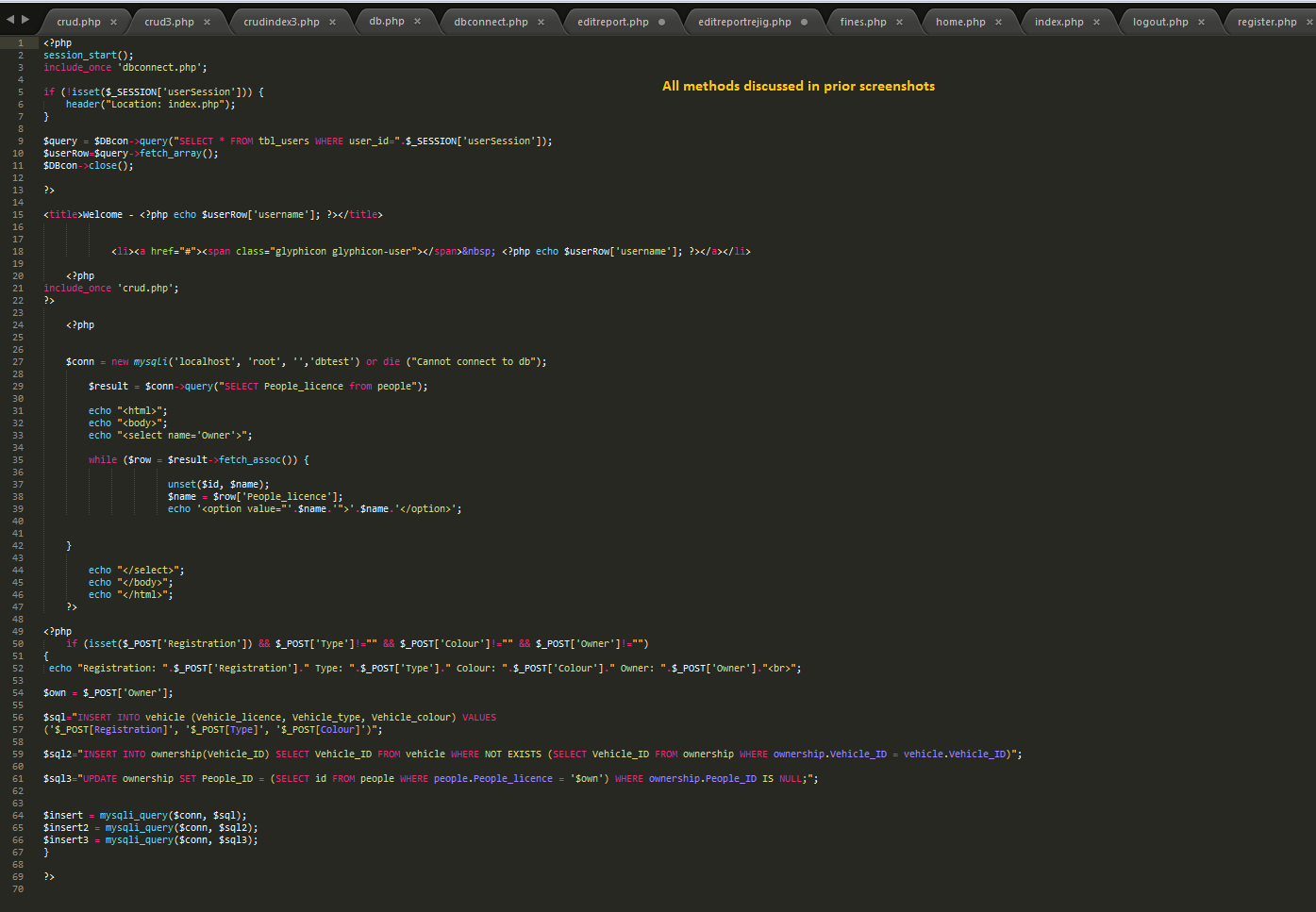
Register.php



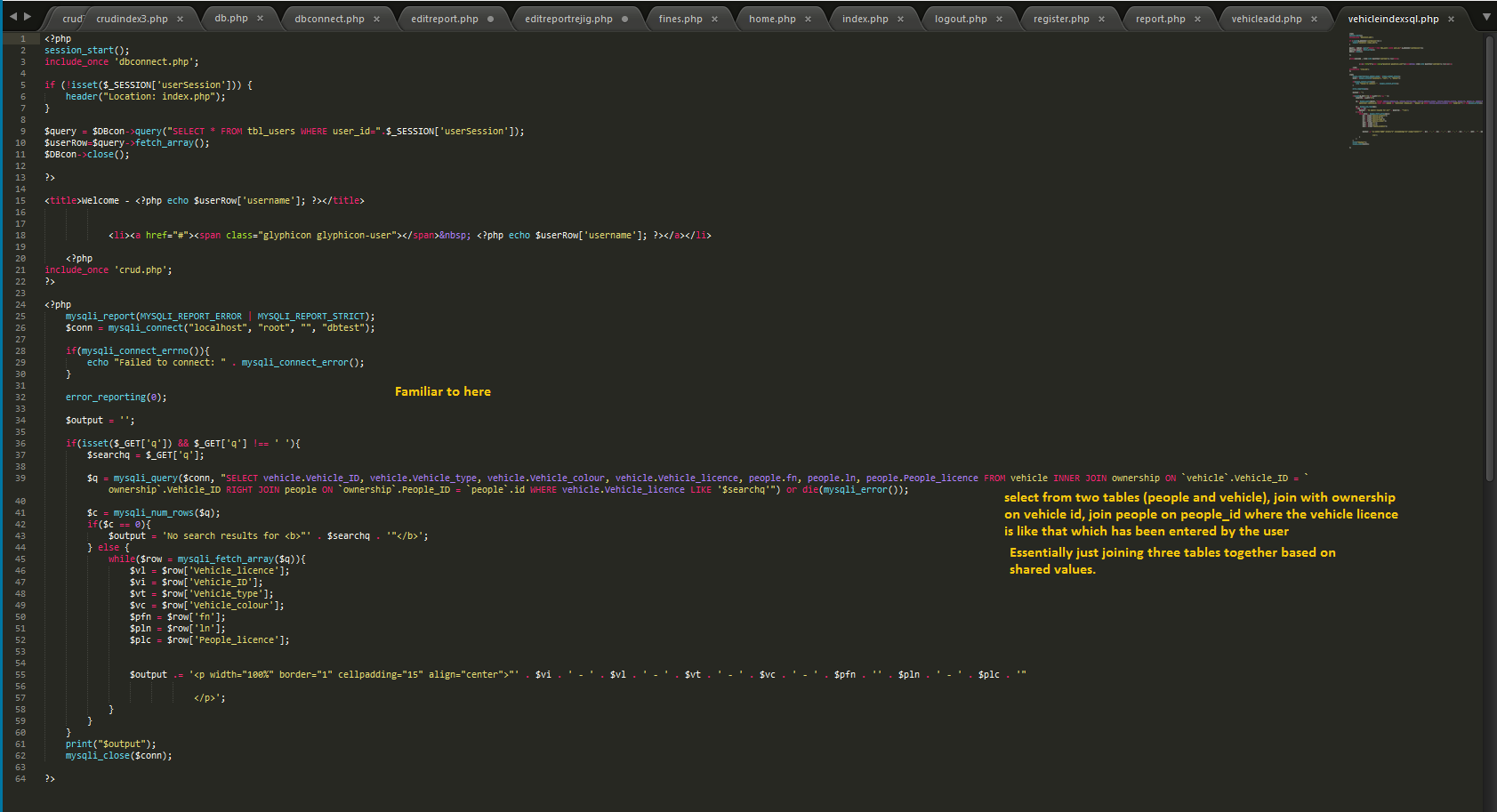
Report.php



Vehicleadd.php



Vehicleindex.php



This covers all of the PHP and SQL used in the project, where possible I have tried to reduce repetition of commentary where the code is identical to that of other php files and have pointed out where this is the case.

­­­