|  |  |  |  |
| --- | --- | --- | --- |
| **CSY3024 Database Technology 2** | | | |
| Date of Issue: | 1st Oct 2016 | Deadline for  Submission: | 23:59 on 15th Jan 2017 |
| Weighing: | **50%** of the Module's assessment | Module Tutor:  Signed: | Dr James Xue |
| sig-small |
| Student ID: | 14438496 | Student Name: | Warren Haskins |
| Video demo link | https://northampton.mediaspace.kaltura.com/media/t/1\_gr5g6cqr# | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | | | | |
| **Assessment Feedback** | | | | | | | |
| Aspect | | | Excellent | Good | Satisfactory | Needs some more work | Needs much more work |
| Database | ERD Model (25%) | |  |  |  |  |  |
| DDL (10%) | |  |  |  |  |  |
| DML (25%) | |  |  |  |  |  |
| Software | CRUD (10%) | |  |  |  |  |  |
| Demonstration (15%) | | |  |  |  |  |  |
| Report quality (15%) | | |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| MARKER’S COMMENTS: | | | | | | |
| **Tutor’s** Signature: |  | Date: |  | Grade: |  |

The University of Northampton’s Policy on Plagiarism & Mitigating Circumstances will be strictly implemented. By submitting this **signed** (typed name in e-submission)assignment, you are asserting that this submission is entirely your own individual work.

DATABASES 3 ASSIGNMENT 1

WARREN HASKINS

14438496

Contents

[1. Introduction 6](#_Toc471394849)

[2. ERD 7](#_Toc471394850)

[3. Table Specification and Entity Explanation 7](#_Toc471394851)

[3.1 clients 7](#_Toc471394852)

[3.2 staff 8](#_Toc471394853)

[3.3 offices 8](#_Toc471394854)

[3.4 office\_staff 8](#_Toc471394855)

[3.5 cars 9](#_Toc471394856)

[3.6 faults 9](#_Toc471394857)

[3.7 meetings 10](#_Toc471394858)

[3.8 lessons 12](#_Toc471394859)

[3.9 interviews 13](#_Toc471394860)

[3.10 tests 13](#_Toc471394861)

[4. Queries 14](#_Toc471394862)

[4.1 The names and the telephone numbers of the managers of each office. 14](#_Toc471394863)

[4.2 The names of all female instructors based in the selected office. 15](#_Toc471394864)

[4.3 The total number of staff at each office. 16](#_Toc471394865)

[4.4 The timetable of appointments for a given instructor next week. 17](#_Toc471394866)

[4.5 The details of interviews conducted by a given instructor. 18](#_Toc471394867)

[4.6 The name and telephone number of the staff(s) who are instructors and over 55 years old. 19](#_Toc471394868)

[4.7 The registration of cars that have had no faults found. 20](#_Toc471394869)

[4.8 The names of clients who passed the driving test in a particular month (e.g., in January 2008). 21](#_Toc471394870)

[4.9 The names of clients who have sat the driving test more than three times and have still not passed. 22](#_Toc471394871)

[4.10 The average number of miles driven during a one-hour lesson. 23](#_Toc471394872)

[5. Other Procedures and functions 24](#_Toc471394873)

[5.1 Check Availability 24](#_Toc471394874)

[5.2 date\_diff function 26](#_Toc471394875)

[5.3 schedule\_test procedure 27](#_Toc471394876)

[5.4 find\_meeting\_id 28](#_Toc471394877)

[6. CRUD GUI 28](#_Toc471394878)

[6.1 GUI 29](#_Toc471394879)

[6.2 Create 29](#_Toc471394880)

[6.3 Query 31](#_Toc471394881)

[6.4 Update 32](#_Toc471394882)

[6.5 Delete 33](#_Toc471394883)

[7. Test Plans 35](#_Toc471394884)

[7.1 GUI Test Plan 35](#_Toc471394885)

[7.2 Database Test Plan 36](#_Toc471394886)

[8 Referencing 39](#_Toc471394887)

[8.1 Sources 39](#_Toc471394888)

[9. Appendix 40](#_Toc471394889)

[9.1 Table Screenshots 40](#_Toc471394890)

[Cars 40](#_Toc471394891)

[Clients 40](#_Toc471394892)

[Faults 40](#_Toc471394893)

[Interviews 40](#_Toc471394894)

[Lessons 41](#_Toc471394895)

[Meetings 41](#_Toc471394896)

[office\_staff 42](#_Toc471394897)

[Offices 42](#_Toc471394898)

[Staff 42](#_Toc471394899)

[tests 42](#_Toc471394900)

[9.2 Database Creation Script – DDL Commands 42](#_Toc471394901)

[9.3 Test Data Script 72](#_Toc471394902)

[9.4 CRUD GUI Code 77](#_Toc471394903)

[9.4.1 header.php 77](#_Toc471394904)

[9.4.2 index.php 78](#_Toc471394905)

[9.4.3 functions.php 79](#_Toc471394906)

[9.4.4 addTable.php 86](#_Toc471394907)

[9.3.5 footer.php 87](#_Toc471394908)

[9.3.6 styles.css 87](#_Toc471394909)

Table of Figures

[Figure 1 Database ERD 7](#_Toc471394910)

[Figure 2 Insert triggers for faults table 10](#_Toc471394911)

[Figure 3 Update triggers for faults table 10](#_Toc471394912)

[Figure 4 Insert triggers for meetings table 11](#_Toc471394913)

[Figure 5 Update triggers for meetings table 11](#_Toc471394914)

[Figure 6 Delete triggers for meetings table 12](#_Toc471394915)

[Figure 7 Insert triggers for Lessons table 12](#_Toc471394916)

[Figure 8 Update triggers for Lessons table 12](#_Toc471394917)

[Figure 9 Insert triggers for Interviews table 13](#_Toc471394918)

[Figure 10 Insert triggers for Interviews table 13](#_Toc471394919)

[Figure 11 Insert triggers for Tests table 14](#_Toc471394920)

[Figure 12 Update triggers for Tests table 14](#_Toc471394921)

[Figure 13 manager names and telephone numbers view 14](#_Toc471394922)

[Figure 14 office\_managers View execution and result 15](#_Toc471394923)

[Figure 15 female\_instructors procedure 15](#_Toc471394924)

[Figure 16 females\_in\_office execution 16](#_Toc471394925)

[Figure 17 females\_in\_office result 16](#_Toc471394926)

[Figure 18 staff count procedure 16](#_Toc471394927)

[Figure 19 total\_staff\_per\_office execution and result 17](#_Toc471394928)

[Figure 20 appointments in next week procedure 17](#_Toc471394929)

[Figure 21 staff\_meetings\_next\_week execution and results 18](#_Toc471394930)

[Figure 22 interview details procedure 18](#_Toc471394931)

[Figure 23 interviews\_conducted execution 19](#_Toc471394932)

[Figure 24 interviews\_conducted result 19](#_Toc471394933)

[Figure 25 staff over 55 view 19](#_Toc471394934)

[Figure 26 instructors\_over\_55 View execution and result 20](#_Toc471394935)

[Figure 27 faultless cars view 20](#_Toc471394936)

[Figure 28 faultless\_cars View execution and result 21](#_Toc471394937)

[Figure 29 clients who passed in a particular month procedure 21](#_Toc471394938)

[Figure 30 passes\_in\_month Procedure execution and result 21](#_Toc471394939)

[Figure 31 triple fail check procedure 22](#_Toc471394940)

[Figure 32 triple\_fail\_check procedure execution and result 23](#_Toc471394941)

[Figure 33 average miles driven during lessons procedure 23](#_Toc471394942)

[Figure 34 avg\_miles View execution and result 24](#_Toc471394943)

[Figure 35 check\_client\_availability procedure 25](#_Toc471394944)

[Figure 36 check\_staff\_availability\_update procedure 26](#_Toc471394945)

[Figure 37 date\_diff function 26](#_Toc471394946)

[Figure 38 schedule\_test procedure 27](#_Toc471394947)

[Figure 39 schedule\_test procedure execution and result 28](#_Toc471394948)

[Figure 40 find\_meeting\_id procedure 28](#_Toc471394949)

[Figure 41 GUI opening state 29](#_Toc471394950)

[Figure 42 GUI add staff state 29](#_Toc471394951)

[Figure 43 GUI add staff example 30](#_Toc471394952)

[Figure 44 submitNewStaff PHP function 30](#_Toc471394953)

[Figure 45 GUI query staff state 31](#_Toc471394954)

[Figure 46 allStaff PHP function 31](#_Toc471394955)

[Figure 47 GUI update staff state 32](#_Toc471394956)

[Figure 48 GUI query post update 33](#_Toc471394957)

[Figure 49 submitChanges PHP function 33](#_Toc471394958)

[Figure 50 GUI post deletion of staff member 34](#_Toc471394959)

[Figure 51 GUI post attempted deletion of staff member with scheduled meetings. 34](#_Toc471394960)

[Figure 52 deleteStaff PHP function 34](#_Toc471394961)

[Figure 53 cars table 40](#_Toc471394962)

[Figure 54 client table 40](#_Toc471394963)

[Figure 55 faults table 40](#_Toc471394964)

[Figure 56 interviews table 40](#_Toc471394965)

[Figure 57 lessons table 41](#_Toc471394966)

[Figure 58 meetings table 41](#_Toc471394967)

[Figure 59 office\_staff table 42](#_Toc471394968)

[Figure 60 offices table 42](#_Toc471394969)

[Figure 61 staff table 42](#_Toc471394970)

[Figure 62 tests table 42](#_Toc471394971)

# Introduction

In order to meet the requirements established in the brief textual analysis and entity relationship modelling techniques were used to find the entities, attributes and relationships necessary to build an ERD of a database that meets all of the requirements.

The database comprises of 10 tables:

* Staff – Stores staff data.
* Clients – Stores client data.
* Offices – Stores office data.
* Office\_staff – Stores information about which office each staff member works at.
* Cars – Stores car data.
* Faults – Stores data about car faults.
* Meetings – Stores scheduling data for each meeting between client and staff.
* Interviews – Stores interview specific data for interview meetings.
* Tests – Stores test specific data.
* Lessons – Stores lesson specific data.

In order to implement all of the required queries a series of views, procedures and functions were created to return the requested data.

Many of the tables also make use of triggers to validate fields. The meeting table uses a trigger to delete data in a child table when deleting a meeting to simplify the process of deleting a scheduled meeting.

To showcase some of the database’s basic functionality and the capacity for fetching data through an external interface a basic web GUI was developed. Through the GUI users can create, read, update and delete rows in the staff table.

# ERD

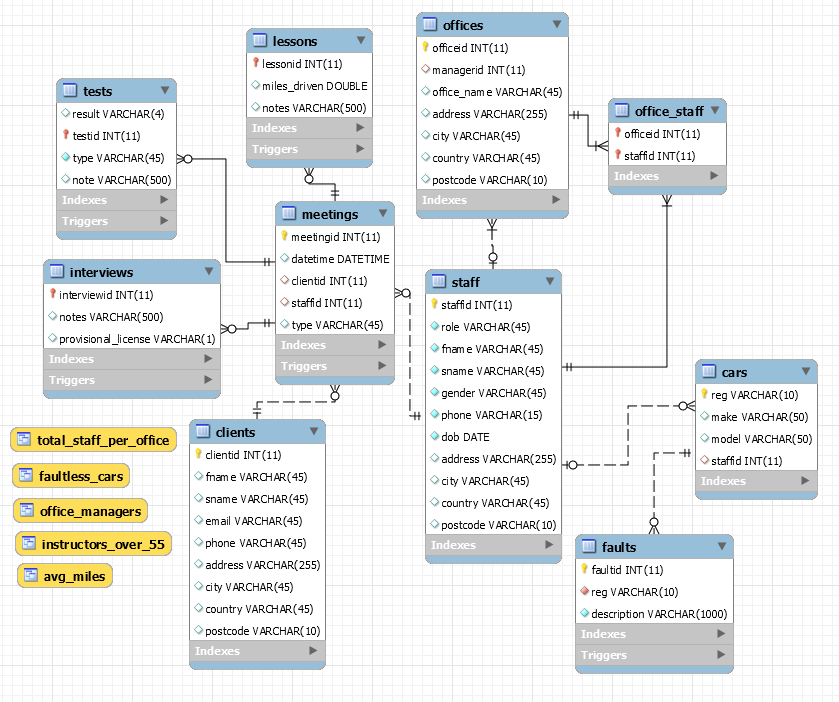


Figure 1 Database ERD

The final ERD comprised of 10 tables and 5 Views. The views are used to hold the queries from a number of the typical transaction queries requested in the brief, the rest of the queries use procedures. All tables, views, and procedures are documented below.

Auto increment is used on the primary key of clients, staff, offices, faults, and meetings to make it easier to add new data to these tables. Without auto increment a new id would have to be made up each time and could clash with an existing id.

# 3. Table Specification and Entity Explanation

|  |  |  |  |
| --- | --- | --- | --- |
| 3.1 clients | | | |
| Attribute | Key (parent) | Datatype | Constraints/Defaults |
| clientid | PK | INT(11) | UNIQUE, NOT NULL, AUTO INCREMENT |
| fname |  | VARCHAR(45) |  |
| sname |  | VARCHAR(45) |  |
| email |  | VARCHAR(45) |  |
| phone |  | VARCHAR(45) |  |
| address |  | VARCHAR(255) |  |
| city |  | VARCHAR(45) |  |
| country |  | VARCHAR(45) | ‘UK’ |
| postcode |  | VARCHAR(10) |  |

Description:

The client table is used to store information about each registered client. The information gathered by the application form when the client first registers is held in this table.

|  |  |  |  |
| --- | --- | --- | --- |
| 3.2 staff | | | |
| Attribute | Key (parent) | Datatype | Constraints/Defaults |
| staffid | PK | INT(11) | UNIQUE, NOT NULL, AUTO INCREMENT |
| role |  | VARCHAR(45) | NOT NULL |
| fname |  | VARCHAR(45) | NOT NULL |
| sname |  | VARCHAR(45) | NOT NULL |
| gender |  | VARCHAR(45) | NOT NULL |
| phone |  | VARCHAR(15) | NOT NULL |
| dob |  | DATE | NOT NULL |
| address |  | VARCHAR(255) |  |
| city |  | VARCHAR(45) |  |
| country |  | VARCHAR(45) | ‘UK’ |
| postcode |  | VARCHAR(10) |  |

Description:

The staff table is used to store information about each staff member. The role attribute holds the current role of the staff member but not whether they are the manager of their office as the brief made it clear that the manager is selected from among those in an existing role, typically senior instructors. The table also contains contact information, an address and the staff members’ DOB. Important information is set to NOT NULL.

|  |  |  |  |
| --- | --- | --- | --- |
| 3.3 offices | | | |
| Attribute | Key(parent) | Datatype | Constraints/Defaults |
| officeid | PK | INT(11) | UNIQUE, NOT NULL, AUTO INCREMENT |
| managerid | FK (staff.staffid) | INT(11) | NOT NULL |
| office\_name |  | VARCHAR(45) |  |
| address |  | VARCHAR(255) |  |
| city |  | VARCHAR(45) |  |
| country |  | VARCHAR(45) | ‘UK’ |
| postcode |  | VARCHAR(10) |  |

Description:

The office table is used to store information about each office, it holds the name and address of the office along with the id of the manager. The managerid column is NOT NULL as each office must have a manager.

Relationships:

The managerid attribute is a foreign key from the staff table referencing the staffid attribute, it is used to hold which staff member is the manager of the office. Each office must have one manager. The relationship is not mandatory for the staff table as not every staff member is a manager.

|  |  |  |  |
| --- | --- | --- | --- |
| 3.4 office\_staff | | | |
| Attribute | Key(parent) | Datatype | Constraints/Defaults |
| officeid | PK, FK(offices.officeid) | INT(11) | NOT NULL |
| staffid | PK, FK(staff.staffid) | INT(11) | UNIQUE, NOT NULL |

The office\_staff table is a link entity used to store which staff members works in each office. The staffid column is unique as each staff member can only work at one office.

Relationships:

The officeid attribute is a foreign key from the offices table referencing the officeid attribute, it is used to hold which office the staff member works at. Each office must have at least one staff member and each staff member belongs to one office. This is one of two values in the compound primary key for this table.

The staffid attribute is the other foreign key that references the staffid attribute in the staff table, it is used to hold the id of a staff member that works at the officeid stored in the same row. This is the second value in the compound primary key for this table.

|  |  |  |  |
| --- | --- | --- | --- |
| 3.5 cars | | | |
| Attribute | Key(parent) | Datatype | Constraints/Defaults |
| reg | PK | VARCHAR(10) | UNIQUE, NOT NULL |
| make |  | VARCHAR(50) |  |
| model |  | VARCHAR(50) |  |
| staffid | FK(staff.staffid) | INT(11) |  |

Description:

The cars table is used to store information about the registered cars and who is using them. The registration plate of the car is used as a primary key since it is a unique identifier for the individual vehicle.

Relationships:

The staffid attribute is a foreign key from the staff table referencing the staffid attribute. The key is used to hold the current user if the car has been assigned a user. Each car does not need to have a current user and a staff member does not necessarily need to have a car assigned to them since there are staff members that are not instructors and only instructors are required to have an assigned car.

|  |  |  |  |
| --- | --- | --- | --- |
| 3.6 faults | | | |
| Attribute | Key | Datatype | Constraints/Defaults |
| faultid | PK | INT(11) | UNIQUE, NOT NULL, AUTO INCREMENT |
| reg | FK(cars.reg) | VARCHAR(10) | NOT NULL |
| description |  | VARCHAR(1000) | NOT NULL |

Description:

The faults table stores information about car faults. When a car has a fault its registration plate and a description of the fault are added to this table. Since a car can have multiple faults the registration is not used as a primary key for this table. All attributes are NOT NULL as they are all necessary for the table to be useful.

Relationships:

The reg attribute is a foreign key from the cars table referencing the reg attribute. The key is used to identify which car has the fault. Each fault needs to have one assigned car but each car does not necessarily have any faults.

Faults Triggers:

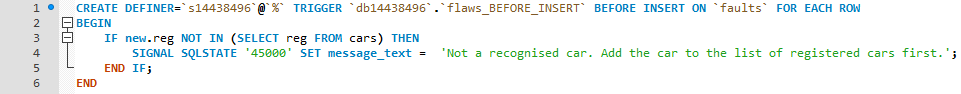


Figure 2 Insert triggers for faults table

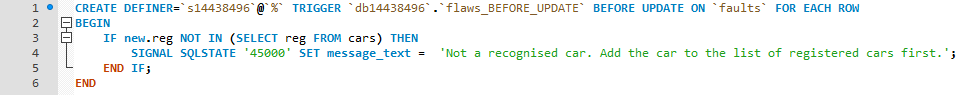


Figure 3 Update triggers for faults table

On update and insert the new value for reg is searched for in the cars table to make sure it exists. If it does not exist an error message informing the user that they must first register the car is displayed.

|  |  |  |  |
| --- | --- | --- | --- |
| 3.7 meetings | | | |
| Attribute | Key | Datatype | Constraints/Defaults |
| meetingid | PK | INT(11) | UNIQUE, NOT NULL, AUTO INCREMENT |
| datetime |  | DATETIME | NOT NULL |
| clientid | FK(clients.clientid) | INT(11) | NOT NULL |
| staffid | FK(staff.staffid) | INT(11) | NOT NULL |
| type |  | VARCHAR(45) | NOT NULL |

Description:

The meetings table is used to record basic information about any meeting scheduled for an instructor or client. It stores information about the individuals involved, the time of the meeting and the type of meeting. The type of meeting determines which table is used to fetch further information about the meeting. All of the attributes in this table are vital so the NOT NULL constraint is employed on each one.

Relationships:

The clientid attribute is a foreign key from the clients table referencing the clientid attribute. This key stores the id of the client that the meeting is scheduled with.

The staffid attribute is a foreign key from the staff table referencing the staffid attribute. This key stores the id of the staff member that the meeting is scheduled with.

Meetings Triggers:

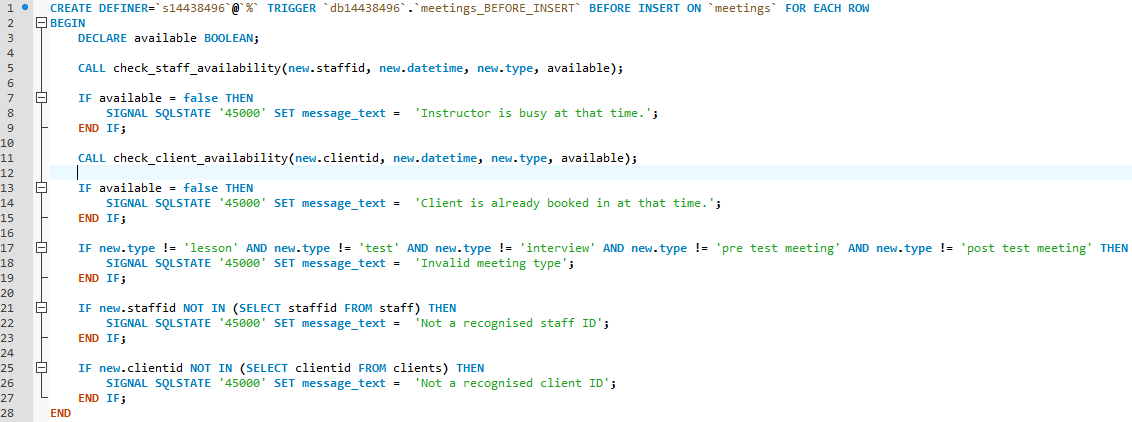


Figure 4 Insert triggers for meetings table

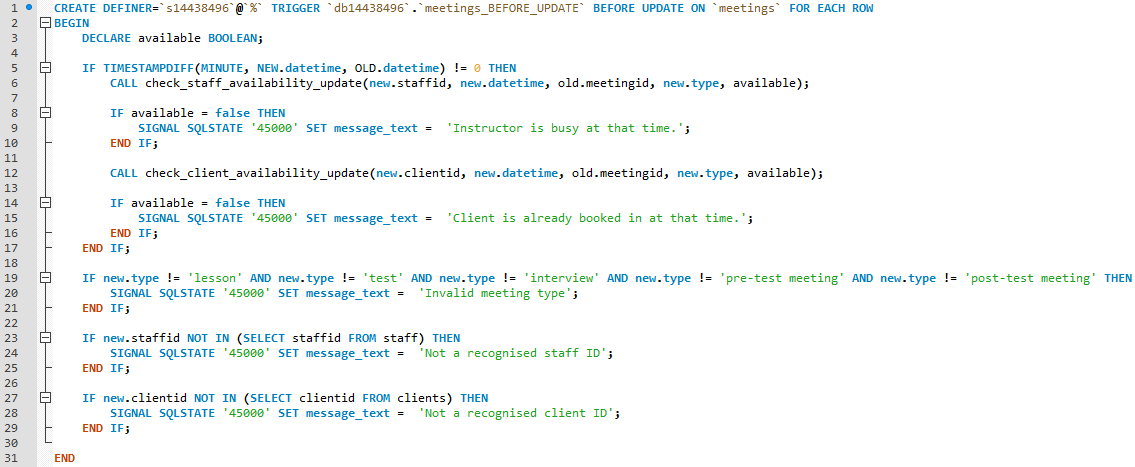


Figure 5 Update triggers for meetings table

The triggers for the meetings table are used to ensure that the client and staff member are both available at the time for the meeting and that both the given client and staff id are from registered individuals. staffid and clientid validation checks aren’t technically necessary since an integrity constraint error will be thrown anyway but this allows clearer feedback to be provided in the error message.

In order to ascertain if a client or staff is available the check\_staff\_availability and check\_client\_availability procedures are used. These procedures are further explained in the ‘Other Procedures’ section of this report, for the sake of this trigger they essentially return true if the client or staff member is available at the given time or false if they are not. The variable ‘available’ is used to store the response. In the update trigger there is an extra if statement that checks if the datetime of the meeting has changed, if it is then relevant procedures check if the new time is free.

Validation is carred out on the type column to ensure that the input is correct and matches one of the meeting types mentioned in the brief. Pre and post-test meetings are used for the brief client – instructor meetings before and after each test.

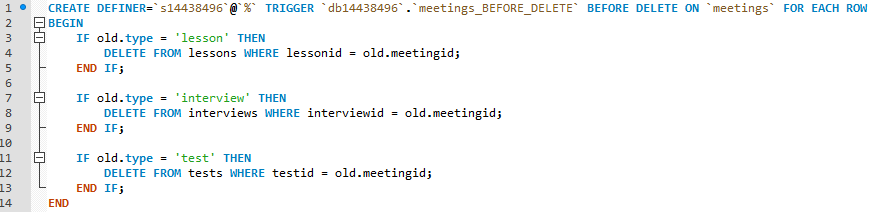


Figure 6 Delete triggers for meetings table

The meetings table also has a trigger that fires when a meeting is deleted. This trigger helps delete the data in the child table associated with each lesson/interview/test meeting. By implementing this trigger deleting scheduled meetings is made much simpler as there is now no need to delete the child tables separately beforehand in order to avoid foreign key integrity errors.

An if statement is used on the type attribute to identify which delete statement should be run.

|  |  |  |  |
| --- | --- | --- | --- |
| 3.8 lessons | | | |
| Attribute | Key | Datatype | Constraints/Defaults |
| lessonid | PK, FK(meetings.meetingid) | INT(11) | UNIQUE, NOT NULL |
| miles\_driven |  | DOUBLE |  |
| notes |  | VARCHAR(500) |  |

Description:

The lessons table is used to record information about an individual lesson. The lessons table allows for lesson specific information to be recorded without cluttering the meetings table with extra, lesson specific, attributes. The rest of the information about the lesson is found in the meetings table.

Relationships:

The lessonid is a foreign key from the meetings table referencing the meetingid. This relationship links the lesson to its parent row in the meetings table. Since the meetingid foreign key is also a unique identifier it used as the primary key for this table. This relationship is necessary as each lesson must have a corresponding meeting. This is an identifying relationship as the foreign key is also the primary key. Not all meetings are lessons so this relationship is not necessary for each row of the meetings table.

Lessons triggers:

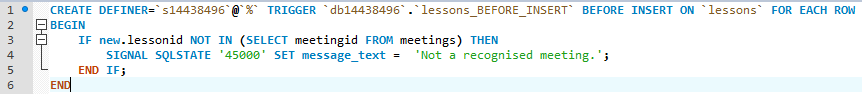


Figure 7 Insert triggers for Lessons table

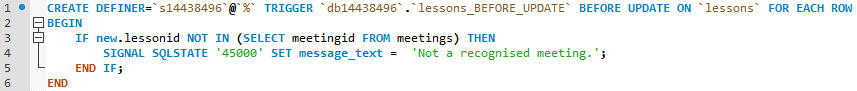


Figure 8 Update triggers for Lessons table

The update and insert triggers on the lessons table are only used for ID validation. If the given ID is not in the meetings table an error message informs the user that the input meeting is not recognised.

|  |  |  |  |
| --- | --- | --- | --- |
| 3.9 interviews | | | |
| Attribute | Key | Datatype | Constraints/Defaults |
| interviewid | PK, FK(meetings.meetingid) | INT(11) | UNIQUE, NOT NULL |
| notes |  | VARCHAR(500) |  |
| provisional\_license |  | VARCHAR(1) |  |

Like the lessons table, the interviews table stores information about an interview. The project brief specifies that during the interview the client’s needs would be assessed and whether they had a provisional license or not would be found so fields are included to hold this information. The provisional license field is limited to accepting either ‘y’ or ‘n’.

Relationships:

The interviewid is a foreign key from the meetings table referencing the meetingid. This relationship links the interview to its parent row in the meetings table. Since the meetingid foreign key is also a unique identifier it used as the primary key for this table. This relationship is necessary as each interview must have a corresponding meeting. This is an identifying relationship as the foreign key is also the primary key. Not all meetings are interviews so this relationship is not necessary for each row of the meetings table.

Interviews Triggers:

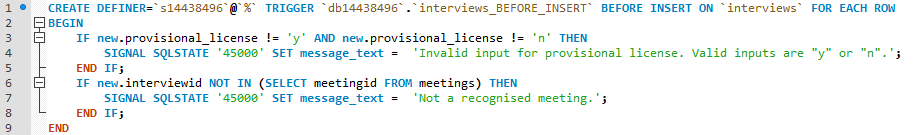


Figure 9 Insert triggers for Interviews table

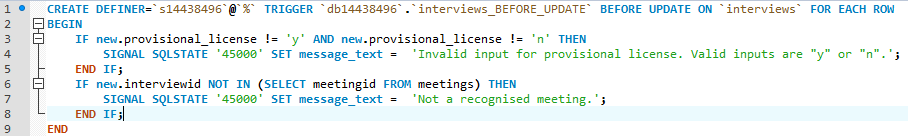


Figure 10 Insert triggers for Interviews table

The triggers for the insert table are used to validate the ID and the provisional license column which stores whether the interviewee owns a provisional license. The accepted parameters are either ‘y’ or ‘n’ since they either have the license or they do not.

|  |  |  |  |
| --- | --- | --- | --- |
| 3.10 tests | | | |
| Attribute | Key | Datatype | Constraints/Defaults |
| testid | PK, FK(meetings.meetingid) | INT(11) | UNIQUE, NOT NULL |
| result |  | VARCHAR(4) |  |
| type |  | VARCHAR(45) | NOT NULL |
| note |  | VARCHAR(500) |  |

Description:

Tests is another table that holds meeting information specific to tests. Each test has a type, either written or driving, and a result so fields are provided to record these. The brief specifies that after a failed test the instructor must note down the reasons for failure so the note attribute is provided for this purpose. The test must have a type so the type column has the NOT NULL constraint.

Relationships:

The testid is a foreign key from the meetings table referencing the meetingid. This relationship links the test to its parent row in the meetings table. Since the meetingid foreign key is also a unique identifier it used as the primary key for this table. This relationship is necessary as each test must have a corresponding meeting. This is an identifying relationship as the foreign key is also the primary key. Not all meetings are tests so this relationship is not necessary for each row of the meetings table.

Tests Triggers:

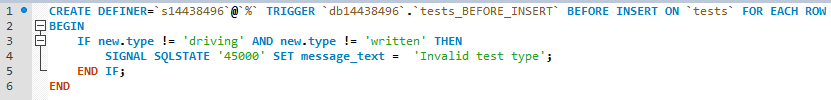


Figure 11 Insert triggers for Tests table

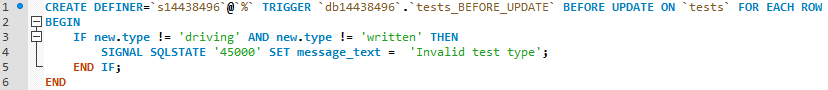


Figure 12 Update triggers for Tests table

# 4. Queries

## 4.1 The names and the telephone numbers of the managers of each office.

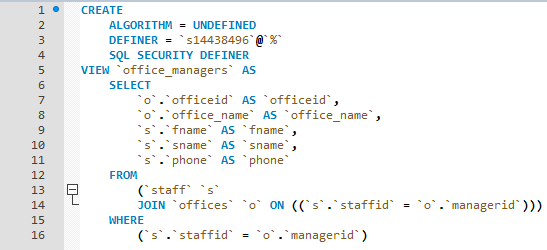


Figure 13 manager names and telephone numbers view

Here a view is used to store the query that retrieves the requested data. The relevant information is stored in the offices and staff tables so they are joined together via the manager id and one where clause is used to narrow down the returned staff to just those that are managers.

Execution and result:

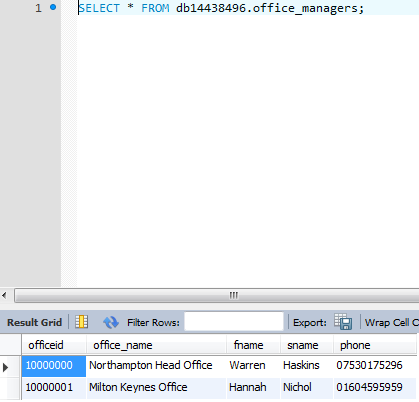


Figure 14 office\_managers View execution and result

## 4.2 The names of all female instructors based in the selected office.

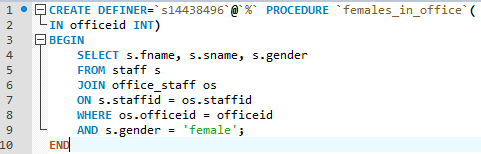


Figure 15 female\_instructors procedure

Here a procedure is used so that the id of the office can be passed in. The staff table stores all of the pertinent information so no join is required. A where clause narrows the returned values down to those that are female and have an officeid that matches the value passed into the procedure.

Execution:

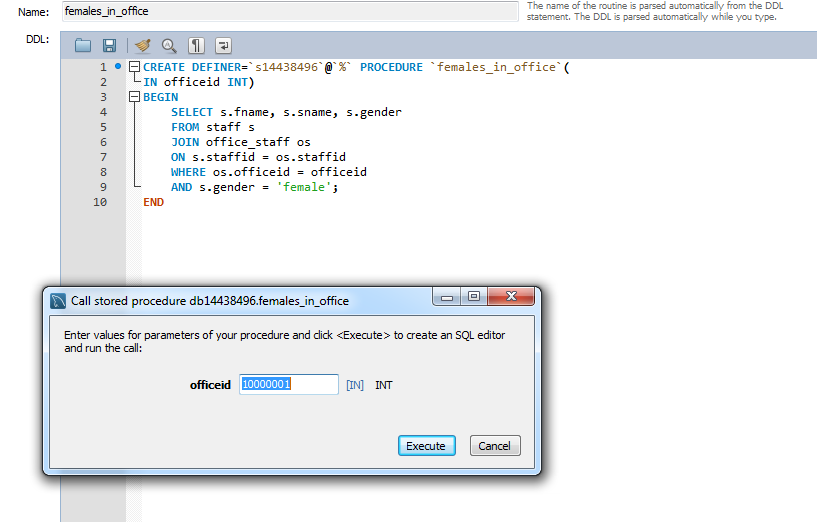


Figure 16 females\_in\_office execution

Result:

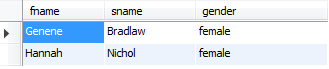


Figure 17 females\_in\_office result

## 4.3 The total number of staff at each office.

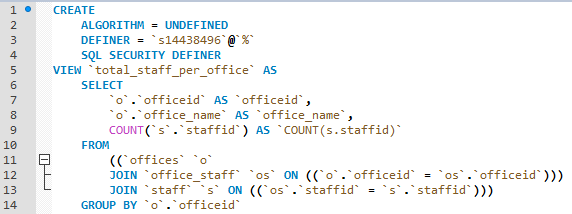


Figure 18 staff count procedure

For this query a view is used as there’s no need to pass in any parameters. The count function is used to get the number of staff at each office. Using a count function means the query can only return one row so the group by clause is implemented to get around this limitation.

Execution and result:

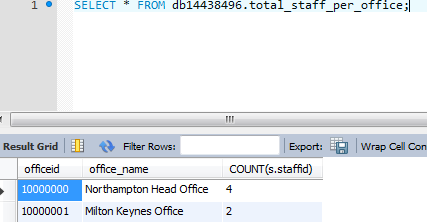


Figure 19 total\_staff\_per\_office execution and result

## 4.4 The timetable of appointments for a given instructor next week.

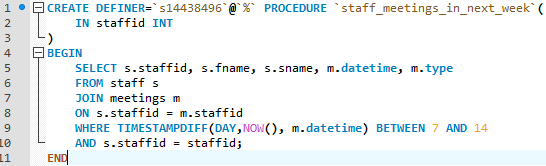


Figure 20 appointments in next week procedure

In order to get all of the appointments for a given instructor next week the TIMESTAMPDIFF function is used in the where clause to narrow down the query to all meetings that occur between 7 and 14 days from the current time. A procedure is used rather than a view so the individual staffid can be passed in and used in the query.

Execution and result ran on the 04/01/2016:

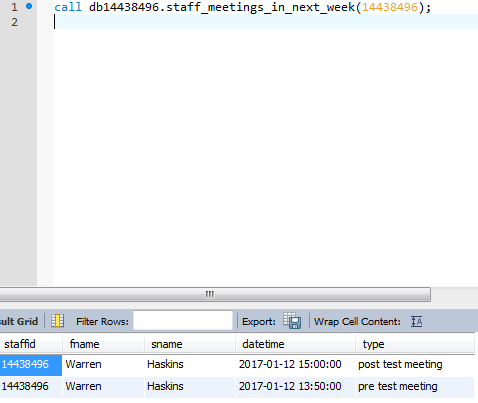


Figure 21 staff\_meetings\_next\_week execution and results

## 4.5 The details of interviews conducted by a given instructor.

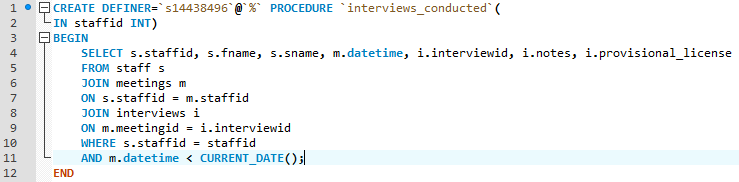


Figure 22 interview details procedure

This procedure takes a staffid parameter to refine the search and then joins together the three tables required to assemble the requested information. A where clause stipulating that the meeting date must be before the current date means that only interviews that have already been conducted are returned and not future planned interviews.

Execution:

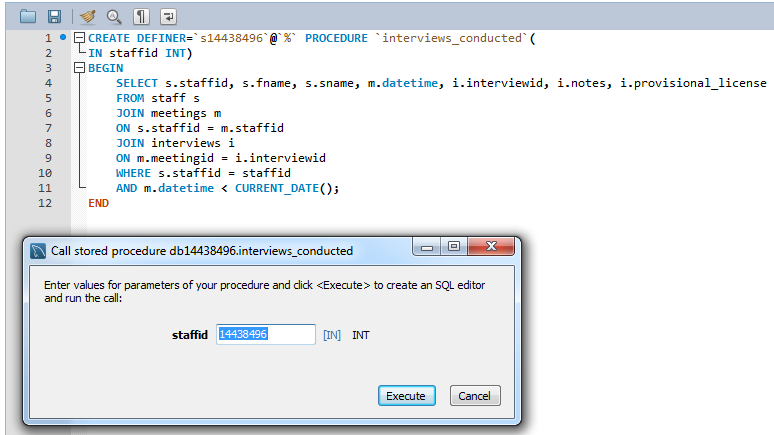


Figure 23 interviews\_conducted execution

Result:



Figure 24 interviews\_conducted result

## 4.6 The name and telephone number of the staff(s) who are instructors and over 55 years old.



Figure 25 staff over 55 view

To find the name and telephone number of the staff who are instructors over 55 years old a view is used as no parameters are required. To work out the age of each staff member a formula is used on the stored DOB of the staff. The formula is the current year minus the year of the staff members DOB, then if the month and day of the current date is before the month and day of the staff’s DOB the ‘<’ operator returns 1 which is subtracted from the previous total, if the month and day are after the DOB month and day a 0 is returned instead so nothing is subtracted.

Execution and result:

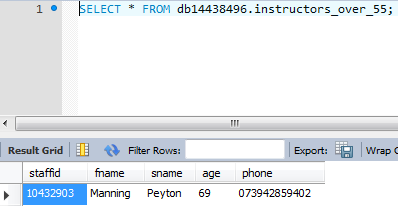


Figure 26 instructors\_over\_55 View execution and result

## 4.7 The registration of cars that have had no faults found.

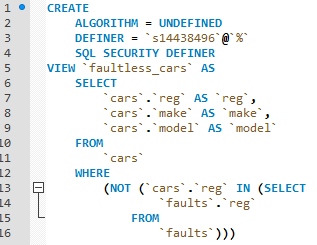


Figure 27 faultless cars view

In order to find all the cars that do not have a registered fault all that is required is to return all of the cars whose registration plate number does not appear in the faults table. This is done using the NOT and IN clauses on a search of the faults table.

Execution and result:

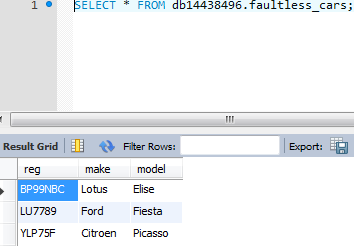


Figure 28 faultless\_cars View execution and result

## 4.8 The names of clients who passed the driving test in a particular month (e.g., in January 2008).

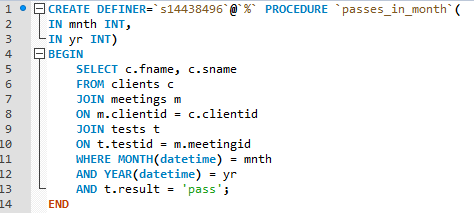


Figure 29 clients who passed in a particular month procedure

To get the clients who passed a driving test in a particular month a procedure that takes two parameters for the year and month is used. The parameters are used in the where clause of a query on the joined clients, meetings and tests tables.

Execution and result:

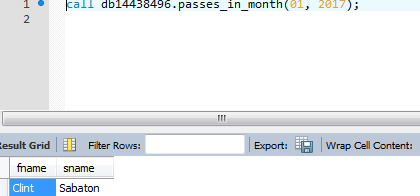
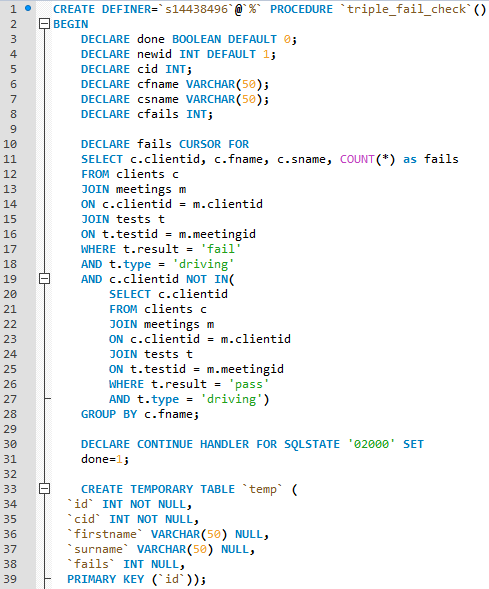


Figure 30 passes\_in\_month Procedure execution and result

## 4.9 The names of clients who have sat the driving test more than three times and have still not passed.



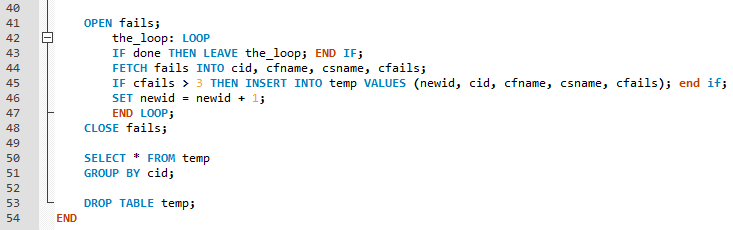


Figure 31 triple fail check procedure

In order to query for clients that have failed there driving test more than three times and still not passed a procedure using a cursor and a temporary table is used. To begin with 6 variables are declared, one to store the Boolean for stopping the cursor, 1 to store the PK of the temporary table and 4 to store the clients details. The cursor joins the clients, meetings and tests tables and queries for all those that have failed a driving test and have never passed a driving test. The query includes the count function to tally up how many fails each client that has never passed has.

The continue handler declaration is used to help break out of the loop if there are no more rows left in the cursor query.

A temporary table is created to hold all the rows from the previous query where the quantity of fails is above 3.

Using a loop, the cursor values are put into the variables and if the number of fails is above 3 each value is inserted into the temporary table. When the loop has finished the temporary table is queried and dropped.

Execution and result:

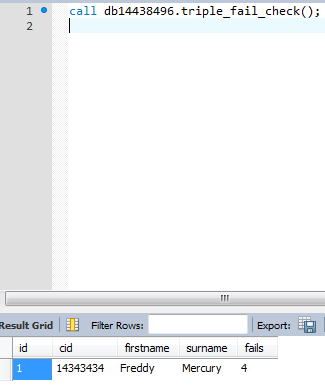


Figure 32 triple\_fail\_check procedure execution and result

## 4.10 The average number of miles driven during a one-hour lesson.

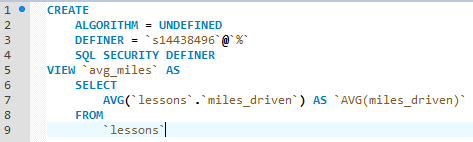


Figure 33 average miles driven during lessons procedure

In order to get the average number of miles driven during a lesson a simple use of the AVG function on the miles\_driven column provides the answer.

Execution and result:

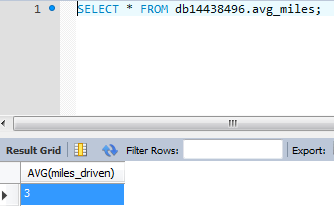


Figure 34 avg\_miles View execution and result

# Other Procedures and functions

This section of the report documents procedures and functions that were created for purposes other than fulfilling the list of required typical transaction in the project brief.

## 5.1 Check Availability

There are 4 separate procedures that are used to check availability in the triggers for the meetings table. There are client and staff variants and insert and update variants. The difference between the client and staff versions is simply which table they search and which ID is passed to them.

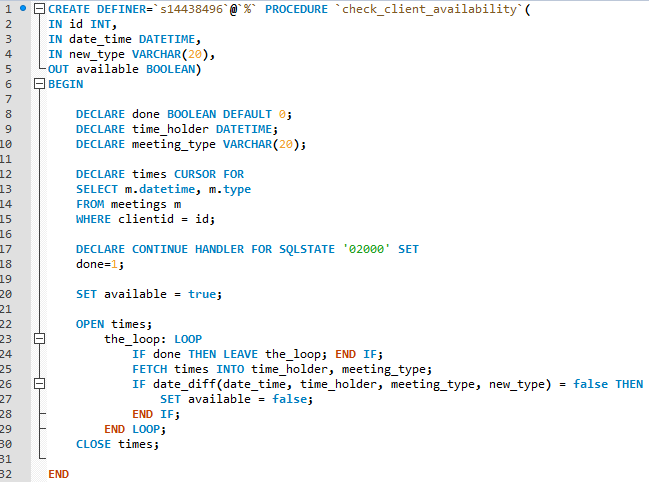


Figure 35 check\_client\_availability procedure

The check\_client\_availability procedure is used by insert triggers to check whether a client has an existing meeting that would clash with the new one. The procedure takes three parameters for the client’s id, the datetime of the new meeting and the type of the new meeting, it outputs a Boolean.

Three variables are declared inside the procedure, one to store a Boolean for exiting the loop used to cycle through the cursor, one to hold the datetime value retrieved from the cursor, and one to hold the meeting type retrieved from the cursor. The cursor contains a query of all the meeting dates and types from the times table which involve the client.

In the loop the cursor’s results are handled by fetching the results into the variables and then passing them to the date\_diff function, this function essentially returns true/false depending on if the two provided datetimes of the provided meeting types would clash. That function is explained in more detail later in the report.

Once the cursor is empty the continue handler is triggered to exit the loop and the cursor is closed.

The staff variant used on table updates can be seen below.

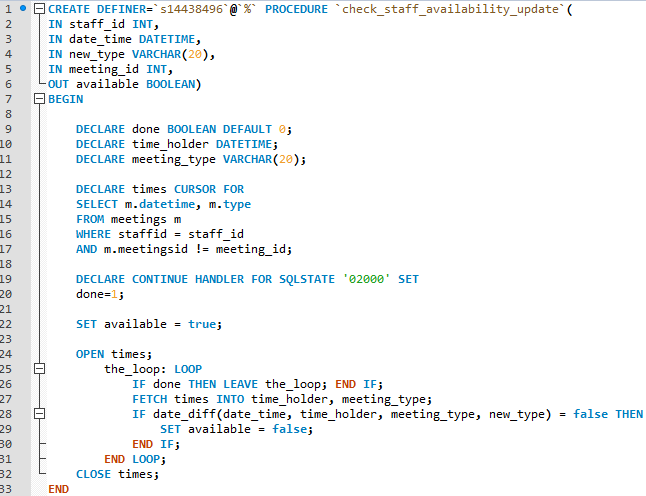


Figure 36 check\_staff\_availability\_update procedure

This procedure is slightly different from the previous one as it has to take an extra parameter in order to make sure the existing row of the meetings table, that the user is trying to edit, does not get factored into the calculation to see If that slot is free. For example, if a user wanted to move a meeting 20 minutes later the previous procedure would reject it as the query would include the existing meeting being edited as a conflicting meeting. This procedure takes the meetingid of the meeting being edited and adds a line to the where clause so that meeting is not returned.

## 5.2 date\_diff function

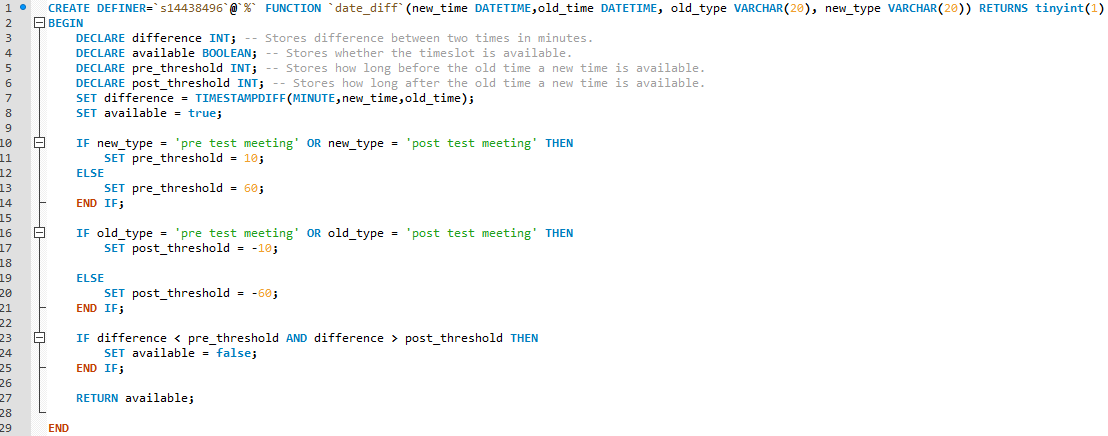


Figure 37 date\_diff function

The date\_diff function is used by the check\_availability functions to work out whether two meetings will clash. It works by taking the datetime and meeting type of each meeting and using the TIMESTAMPDIFF MySQL function to get the difference between the two in minutes.

Most meetings are assigned one hour slots as the brief only specifies the length of time that lessons take. Since the brief states that the instructor must be present at the beginning and end of each test the pre and post-test meeting types are provided to timetable these required meetings and 10 minutes is allocated for each one.

In order to see if the proposed meeting would fall outside the time of the existing meeting two thresholds are established depending on the meeting type. The pre threshold holds how long before the old meeting the new meeting can begin and the post threshold holds how long after the old meeting has begun another meeting can be held. The thresholds are assigned based on what the old and new meeting types are. If the difference between the times falls within the two thresholds the timeslot is not available and the function returns false to indicate this.

## 5.3 schedule\_test procedure

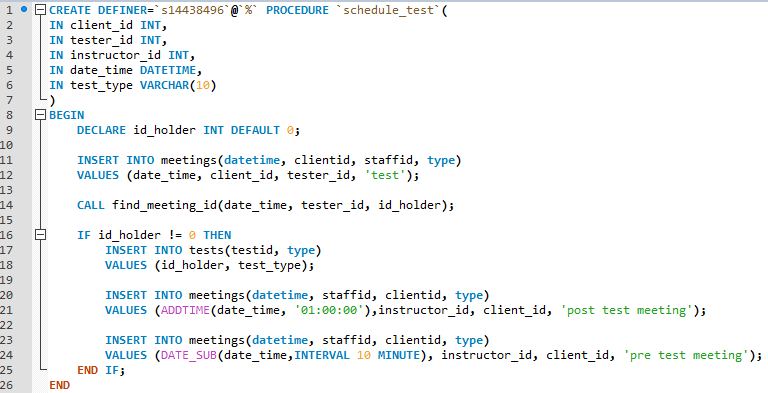


Figure 38 schedule\_test procedure

The schedule\_test procedure is a procedure developed to reduce the hassle of scheduling new tests since each new test requires also creating two additional meetings between the client and an instructor before and after the test. This procedure takes 5 parameters for the client, tester, instructor, scheduled date and the type of test and uses them to populate the appropriate insert statements. In order to get the automatically generated meeting id that is created in the first insert statement the find\_meeting\_id procedure is used, this procedure returns the id of the meeting at the specified time with the specified instructor. If the meeting causes a schedule clash the if statement prevents any of the other meetings being added.

The ADDTIME and DATE\_SUB MySQL functions are used to set the proper times for the pre and post-test meetings.

Execution:

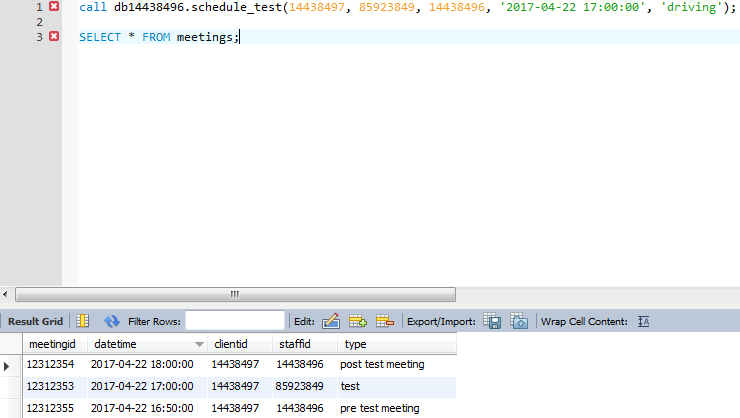


Figure 39 schedule\_test procedure execution and result

## 5.4 find\_meeting\_id

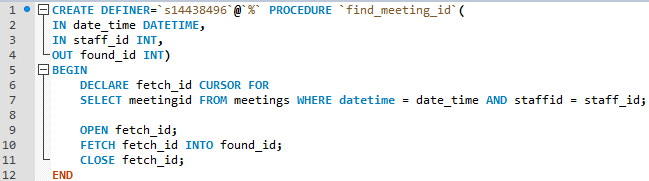


Figure 40 find\_meeting\_id procedure

The find\_meeting\_id procedure is used by the schedule\_test procedure to find the meetingid of the meeting scheduled at a given time with a given instructor. Since each instructor can only be in one place at a time the combination of instructorid and datetime is a unique identifier for a meeting. The procedure uses a cursor to find the meetingid and fetch it into the returned variable.

# CRUD GUI

As part of the assignment a basic GUI was developed to showcase some of the basic database functionality. The GUI developed is a webpage programmed using PHP, HTML and CSS.

## 6.1 GUI

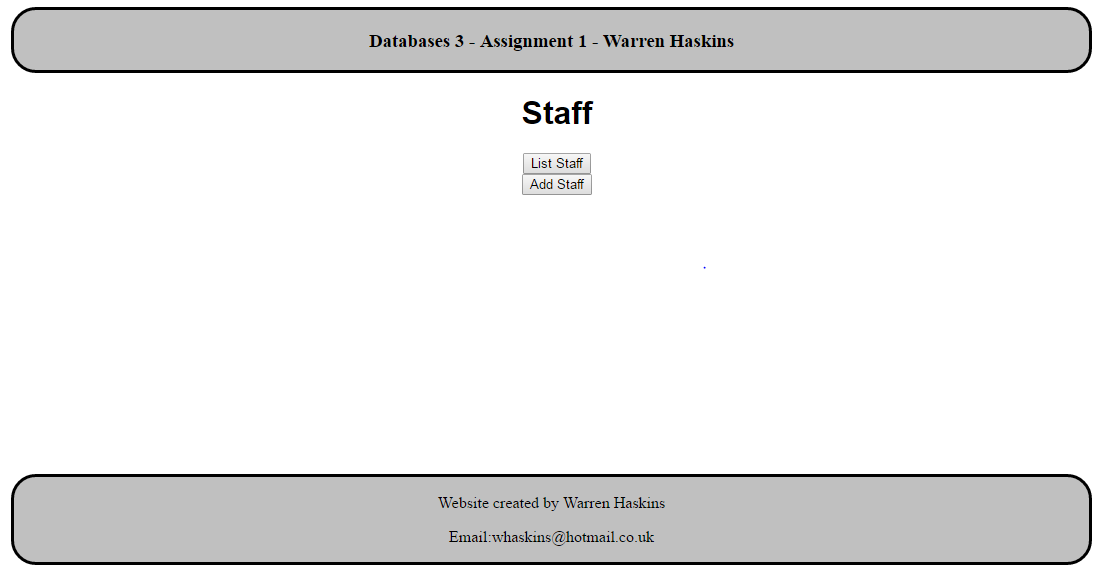


Figure 41 GUI opening state

In its opening state the GUI provides options to either list the staff or add a new staff member.

## 6.2 Create

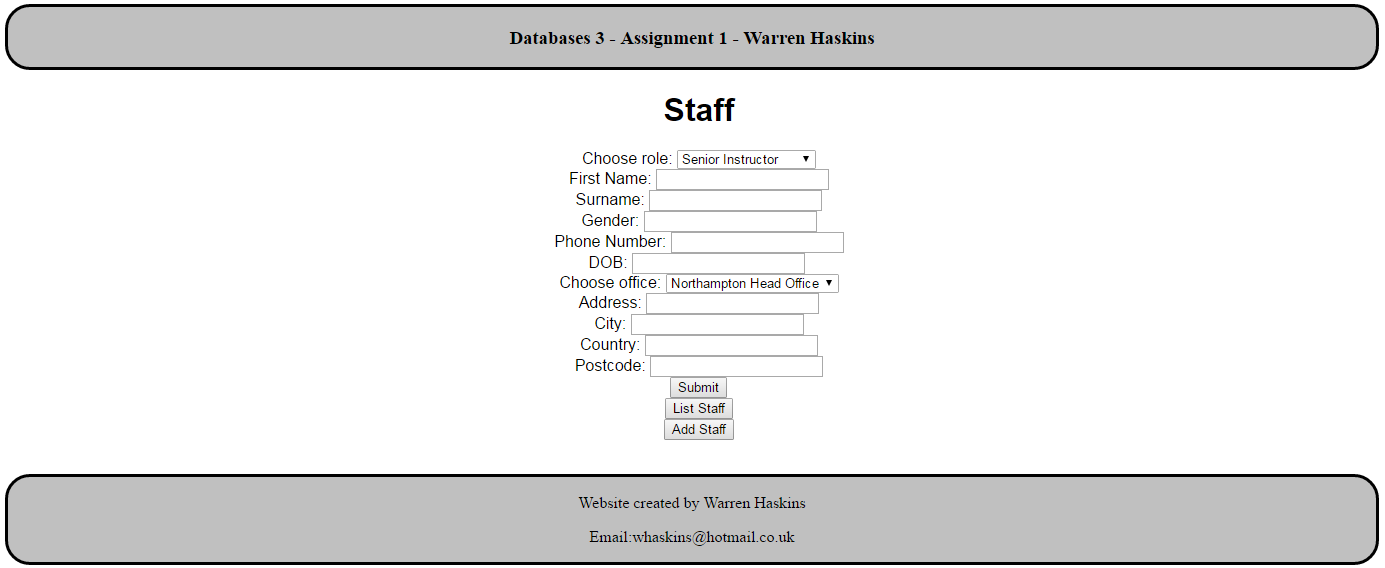


Figure 42 GUI add staff state

When the add staff button is selected the GUI transitions to the above screen. A dropdown menu stores all the valid options for a new staff member’s role and text fields are used to allow inputs into the other columns. A query of the offices table is used to populate the ‘Choose office’ dropdown menu with a list of the offices in the database. When submitted the new staff member is added to the database if the inputs are valid.

For showcase purposes a new staff member will be added called Rick Hanshaw.

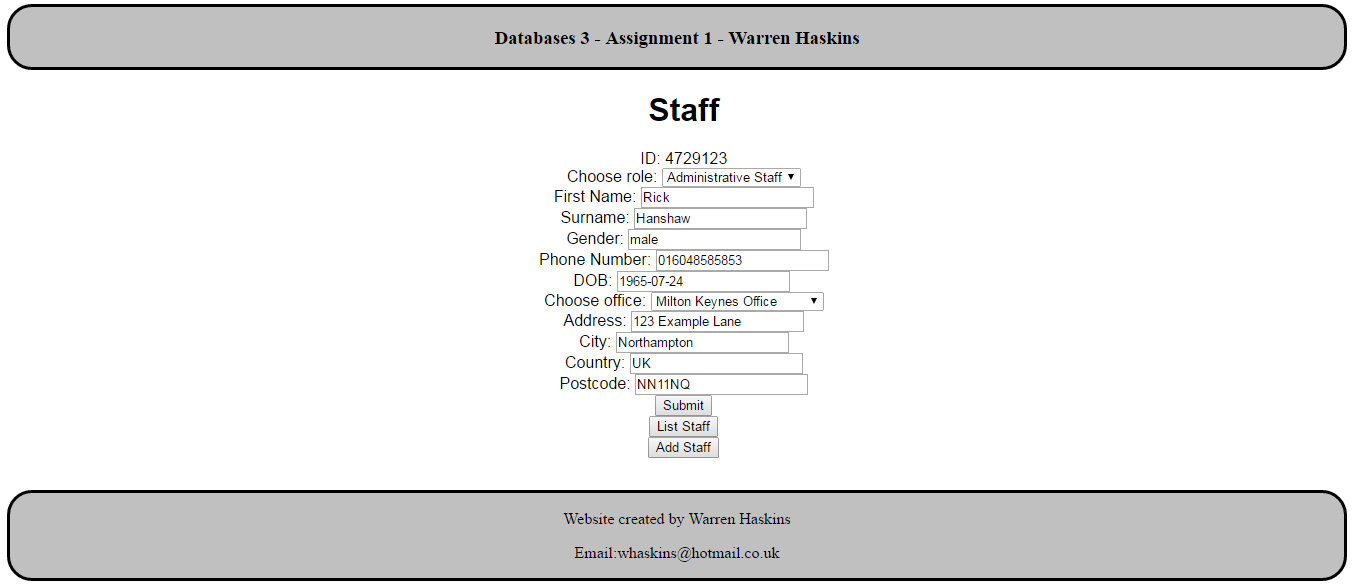


Figure 43 GUI add staff example

Once all the fields are populated selecting the submit button will add Rick to the database. The function that does this is below:

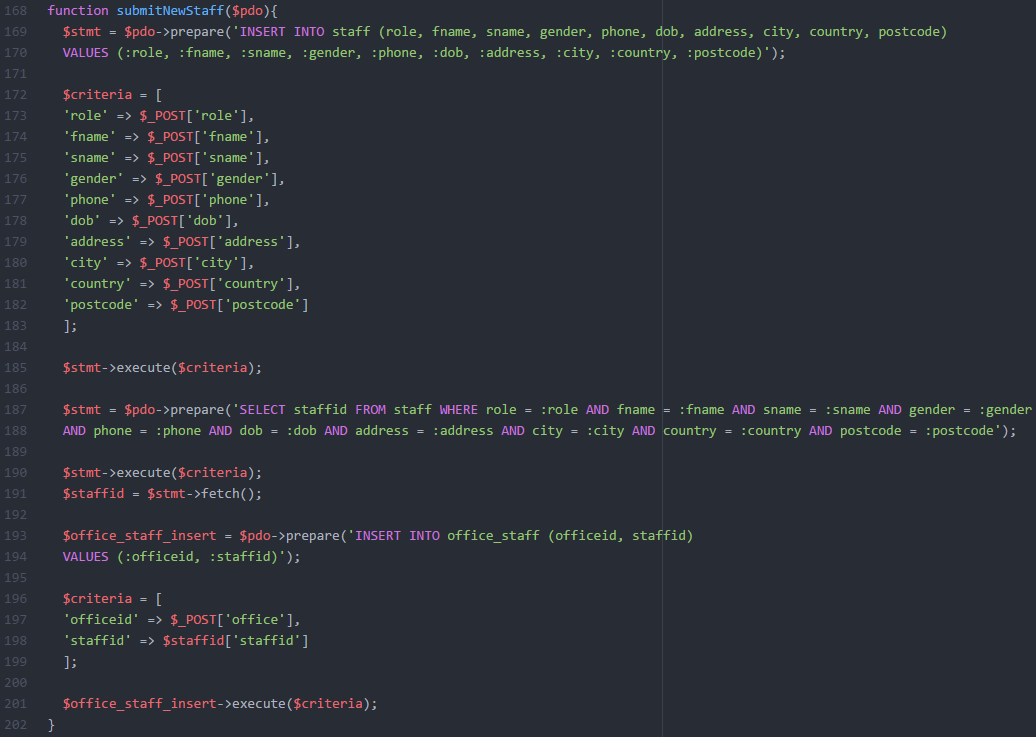


Figure 44 submitNewStaff PHP function

This function uses a prepared insert statement to hold the query. The data from the form submission is added to the variables in the criteria array and upon execution the prepared statement is filled with the values from the criteria array and sent to the database. Using prepared statements helps to prevent SQL injection, where a user might write SQL into a text field to try and tamper with the database. Another prepared statement is used to obtain the newly generated staffid from the staff table and then this is added to the office\_staff table along with the id of the office that was selected.

## 6.3 Query

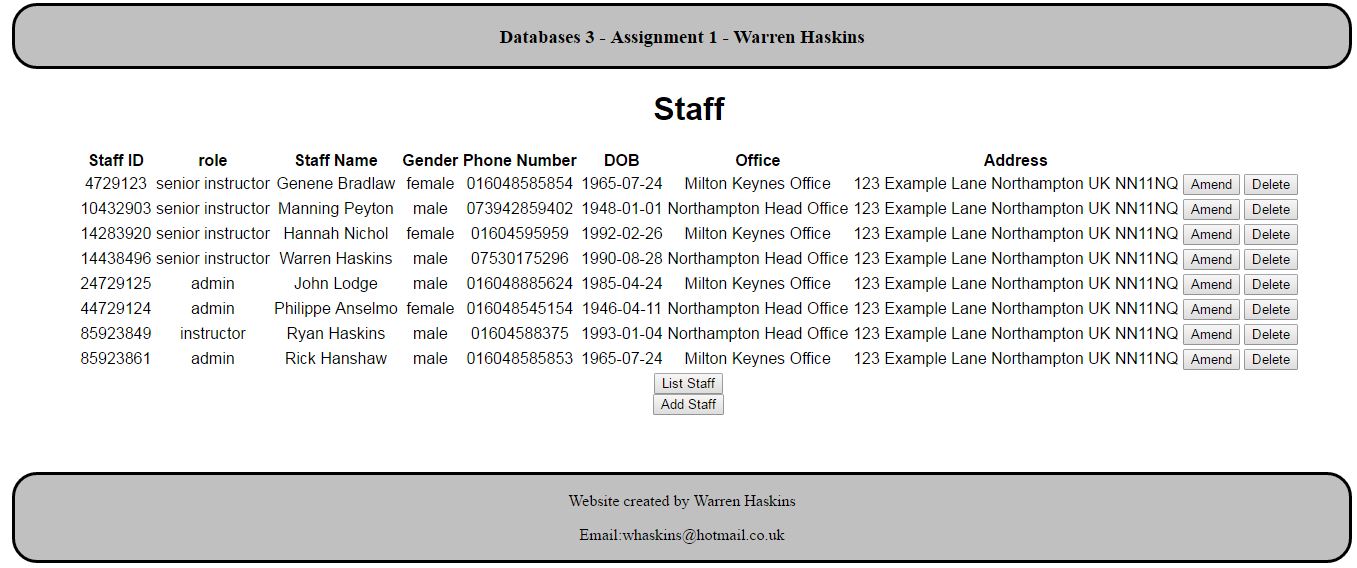


Figure 45 GUI query staff state

When the list staff button is selected the GUI is populated with a list of all of the staff members stored in the staff table. The relevant data is displayed in a table format. The name of the office the staff member works at is also retrieved from the offices table by using a query that joins the staff, office\_staff and offices tables together. The newly added staff member Rick Hanshaw can be seen at the bottom of the list.

The PHP code for querying the staff table can be seen here:

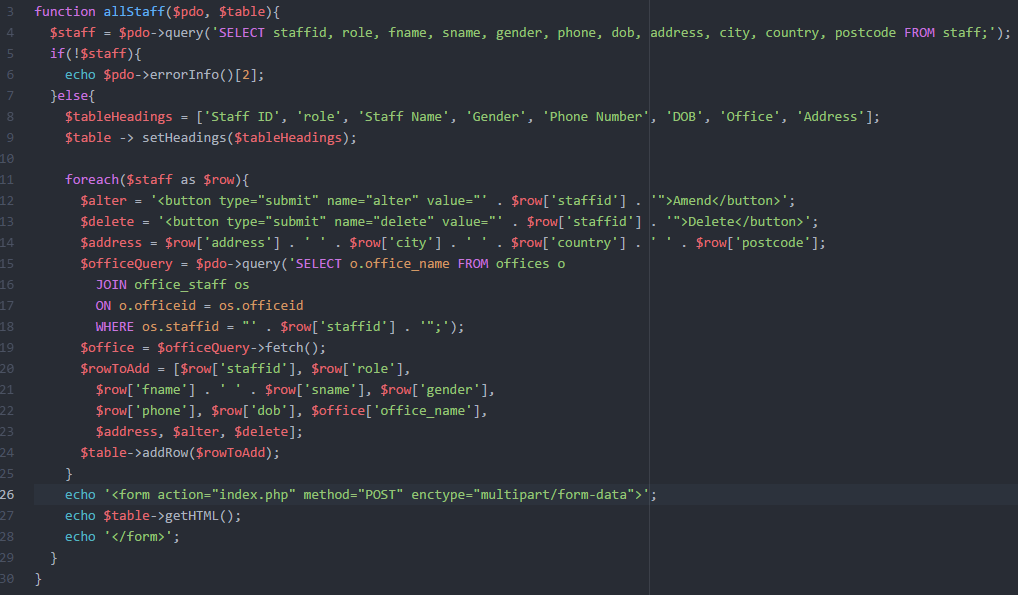


Figure 46 allStaff PHP function

The function uses a query to fetch every staff member from the staff table. It then passes a set of table headings to the table class which was instantiated earlier in the code. The table class essentially just takes an array for the head of the table and an array for each row of the table and then, when prompted, returns the html code required to create the table.

For each row returned by the staff query the html for creating a submit and delete button with their staffid as their value is added to a variable. The name of the office that the staff member works at is found by using another query on the office\_staff table joined to the offices table, the result is stored in the $office variable. At this point the program adds all of the staff’s details into an array along with the variables storing the office name and the two variables storing the buttons. This array is passed to the table class as a row.

Once the for loop has finished a form is created and inside it the html for the table is placed by calling the getHTML function of the table class.

## 6.4 Update

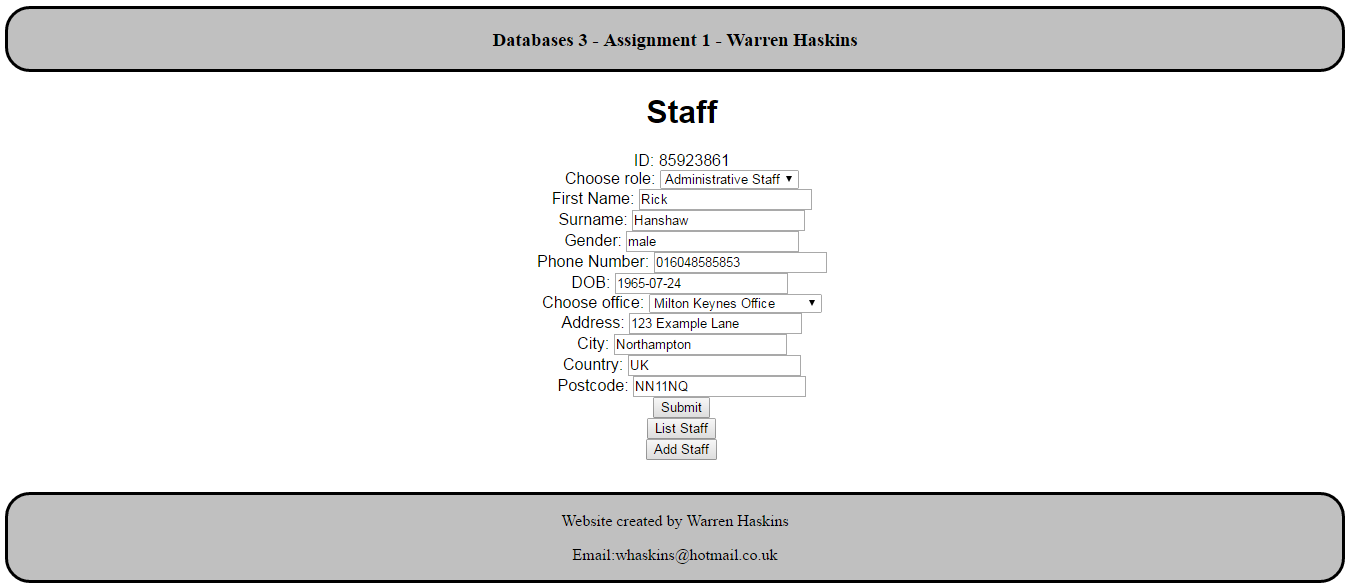


Figure 47 GUI update staff state

When the amend button next to a displayed member of staff is selected the GUI takes the user to the amend staff page where the selected staff member’s details are entered into text fields. From here the user can make changes to any field they like, except for the ID, and submit them with the submit button. Once the submit button is selected the GUI navigates to the list staff page to show the changes have been successful. For example, if we change Rick’s city to Edinburgh this change will be recorded in the database and shown in the GUI:

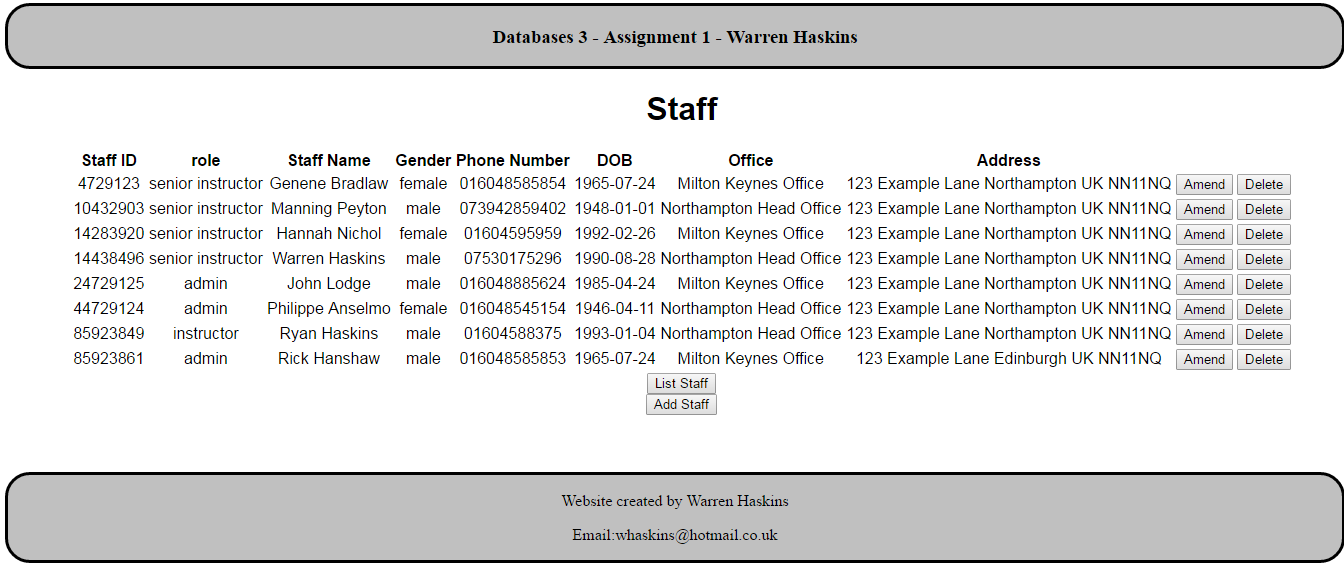


Figure 48 GUI query post update

The PHP function used to update staff members is:

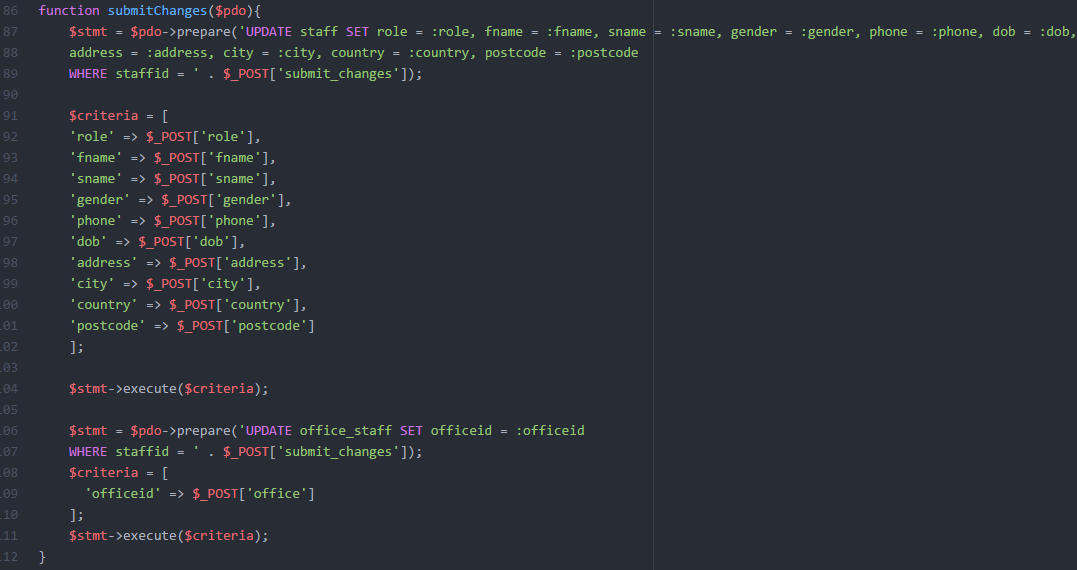


Figure 49 submitChanges PHP function

This function works in the same way as the previous function but uses update statements rather than insert statements. There’s no need to run the extra query to find the id because since a staff member is being updated we already know their id.

## 6.5 Delete

Selecting the delete button next to any staff member in the table will remove them from the database, deleting them from both the staff table and the office\_staff table. Once the button is selected the page refreshes to show that the staff member has been removed. This button will not function for any staff member that has a meeting scheduled and an error message is displayed, the meeting will need to be deleted first. Rick Hanshaw has no meetings to we can delete him safely from the database with no concerns.

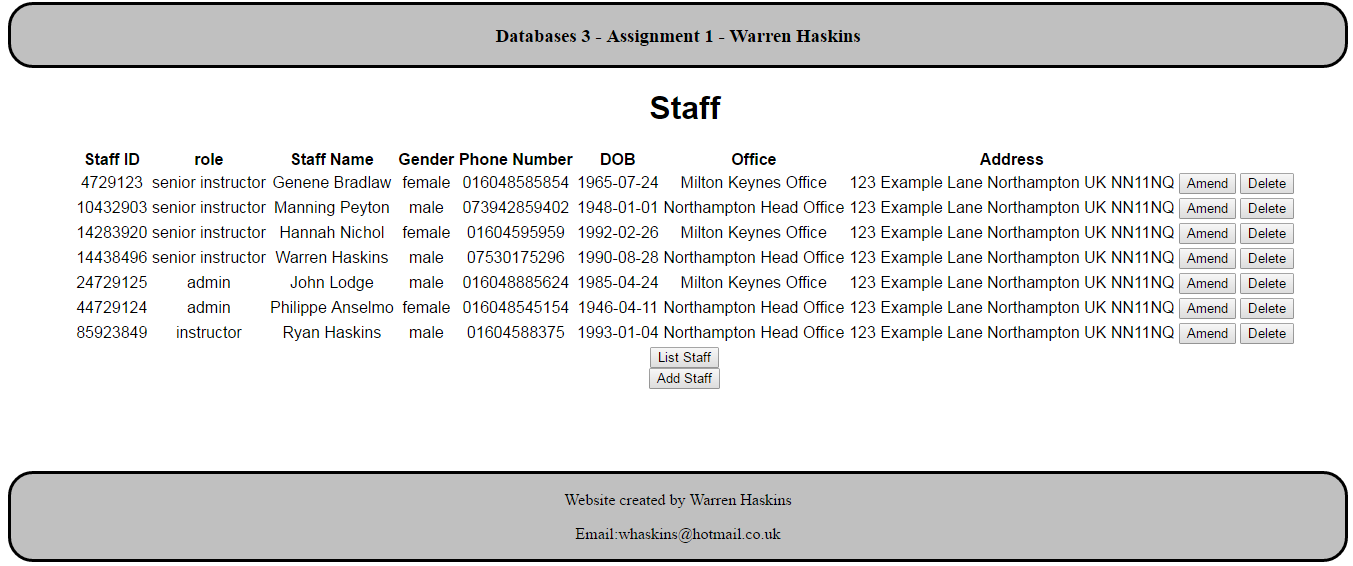


Figure 50 GUI post deletion of staff member

Below can be seen the error message displayed when attempting to delete a staff member who still has meetings scheduled:

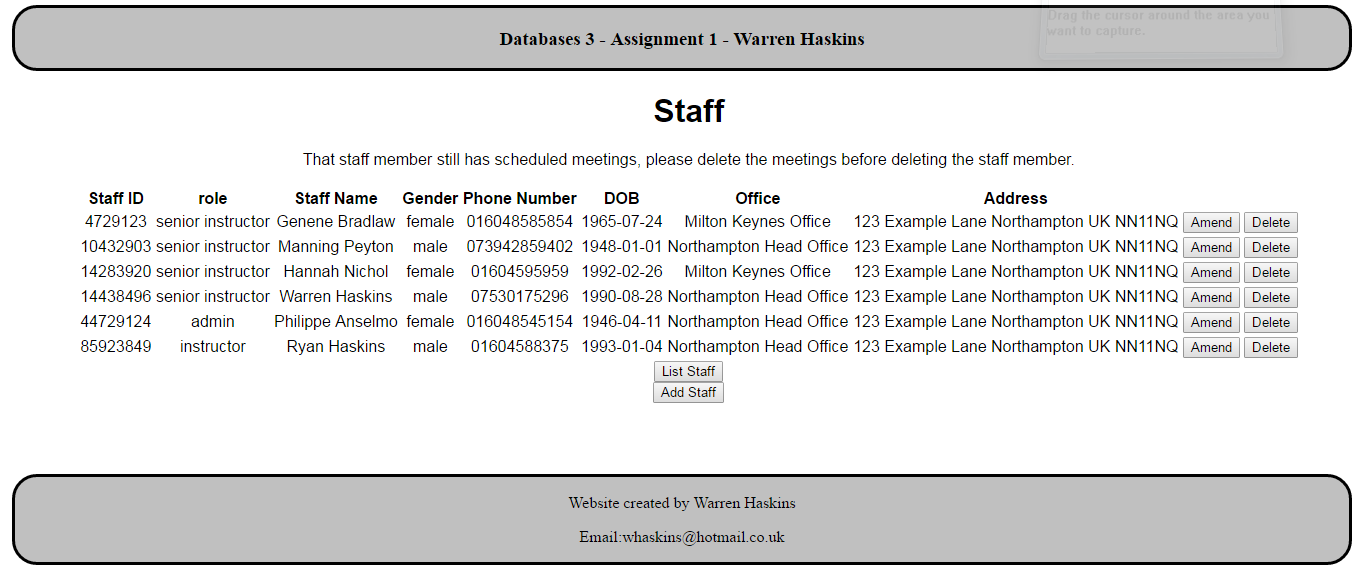


Figure 51 GUI post attempted deletion of staff member with scheduled meetings.

The PHP code that handles deleting staff can be seen here:

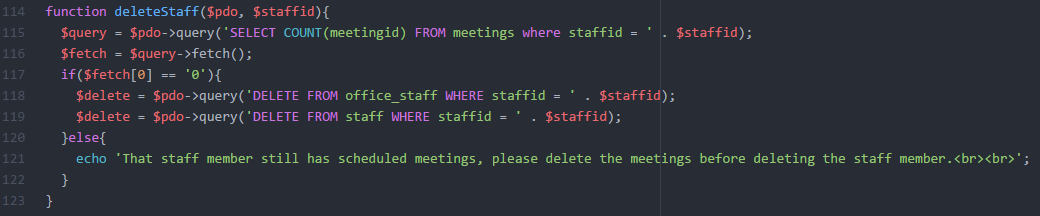


Figure 52 deleteStaff PHP function

This function simply takes the staff member’s id and uses it to query the meeting table, if nothing is returned they are deleted from the office\_staff table and staff table. If they still have meetings scheduled the function prints out the error message.

# 7. Test Plans

## 7.1 GUI Test Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Element being tested | Action performed | Expected Outcome | Actual Outcome |
| List Staff button | Button pressed | All rows in the staff table presented in a table format with office name and amend/delete buttons. | As expected |
| Add staff button | Button pressed | GUI should display a dropdown menu for selecting a role and an office along with textfields for all staff table attributes, except for the staffid. A submit button should appear beneath. | As expected |
| Add a new staff member | Populate all fields and press the submit button. | GUI should display all rows in the staff table including the newly added staff member. New staff member should also be added to the database. | As expected |
| Amend button | Press Amend button next to a staff member | GUI should display populated textfields and dropdown menus for each attribute in the selected staff members row from the staff table. The id should be displayed above the editable fields and a submit button should appear below. | As expected |
| Submit amendments | Alter each fields contents and press the submit button. | GUI should display all rows in the staff table including the changes to the specific staff member. Changes should be reflected in the database. | As expected |
| Delete button on staff member with empty schedule | Press delete button next to a staff member with no meetings. | GUI should display all rows in the staff table. Staff member deleted should be removed from the staff table and thus not displayed. | As expected |
| Delete button on staff member with meetings scheduled | Press delete button next to a staff member with a meeting. | GUI should display all rows in the staff table. Staff member should not be deleted and GUI should display a message informing user of the reason it was not deleted. | As expected |

## Database Test Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Element being tested | Action performed | Expected Outcome | Actual Outcome |
| Database Creation Script | Creation script run on database. | Database populated with entities, attributes and relationships present in the ERD as well as all procedures, views, triggers and functions implemented. | As expected. |
| Test data script | Test data script is run on the database. | Database tables populated with test data. | As expected |
| avg\_miles view | Query the view. | a table containing the average miles driven throughout every driving lesson. | As expected |
| faultless\_cars view | Query the view. | A table containing every car in the cars table that does not have an associated fault in the faults table. | As expected |
| instructors\_over\_55 view | Query the view. | A table containing every instructor over the age of 55. Should not return non instructors over 55. | As expected |
| office\_managers view | Query the view. | A table containing the name and phone number of each manager along with the office they are manager of should be returned. | As expected |
| total\_staff\_per  \_office view | Query the view. | A table containing the office and the number of staff in that office should be returned. | As expected |
| check\_client  \_availability procedure | Pass the procedure a valid id, datetime and type twice. In the first instance pass a free time and on the second pass a time that will class with another. | The procedure should return true in the first instance and false in the second. | As expected |
| check\_client\_  availability\_update procedure | Pass the procedure a valid id, datetime, meetingid and type three times. In the first instance pass a free time and on the second pass a time that will clash with another.  On the third pass a time that will clash with the time of the passed meeting. | The procedure should return true in the first and third instance and false in the second. | Error caused by typo in the where clause.  Fixed typo and reran test. Results as expected. |
| check\_staff\_  availability procedure | Pass the procedure a valid id, datetime and type twice. In the first instance pass a free time and on the second pass a time that will class with another. | The procedure should return try in the first instance and false in the second. | As expected. |
| check\_staff\_  availability\_update procedure | Pass the procedure a valid id, datetime, meetingid and type three times. In the first instance pass a free time and on the second pass a time that will clash with another.  On the third pass a time that will clash with the time of the passed meeting. | The procedure should return true in the first and third instance and false in the second. | Error caused by typo in the where clause.  Fixed typo and reran test. Results as expected. |
| females\_in\_office  procedure | Pass the procedure a valid officeid | The procedure should return a table with all of the female staff from the specified office. | As expected |
| find\_meeting\_id procedure | Run the schedule\_test procedure as it uses this procedure to function. | If the schedule\_test procedure works as intended then this procedure will work as intended. | As expected |
| interviews\_conducted procedure | Pass the procedure a valid staffid. | The procedure should return all interviews conducted by the staff member passed to it. It should not return interviews scheduled for the future but not yet conducted. | As expected |
| passes\_in\_month | Pass the procedure a valid month and year integer. | The procedure should return a table of all clients that passed a driving test in the specified month. | As expected |
| schedule\_test procedure | Pass the procedure the ids for a client, instructor and tester along with a datetime and test type. | If all participants are free at the scheduled time a test will be scheduled between the client and the tester of the type specified and at the time specified. Two further meetings will be scheduled immediately before and after the test between the client and the instructor. | As expected |
| staff\_meetings\_in  \_next\_week procedure | Pass a valid staffid to the procedure | the procedure should return a table of all of the staffs meetings in the next week. | As expected |
| triple\_fail\_check procedure | Run the procedure | The procedure should return a table of all individuals who have failed their driving test more than three times and have not passed. | As expected |
| date\_diff function | Run the availability procedures as this function is working if they are working. | The function will help the procedure to successfully identify whether the timeslot is available for the meeting of that type. | As expected |
| faults\_BEFORE\_INSERT and faults\_BEFORE\_UPDATE  triggers | Try and add a fault with a car reg that is not in the cars table. | The trigger will fire and display an error message informing the user that the car isn’t recognised. | As expected |
| interviews\_BEFORE \_INSERT and interviews\_BEFORE \_UPDATE triggers | Try and change an interviews provisional\_license column to something other than y or n.  Try to change the interviewid to an id that is not in the meetings table | For the first trigger an error message should be triggered telling the user that they need to put either Y or N in that column.  The second trigger should inform the user that the meeting is not recognised if the meeting id is wrong. | As expected |
| lessons\_BEFORE \_INSERT and lessons\_BEFORE \_UPDATE triggers | Try and add a new lesson with a lessonid that does not match a meetingsid in the meetings table. | An error message should be displayed saying that the meeting is not recognised. | As expected |
| meetings\_BEFORE\_INSERT | Try and add a meeting where the client has a schedule clash.  Try and add a meeting where the instructor has a schedule clash.  Try and add a meeting with an invalid meeting type parameter.  Try and add a meeting with invalid clientid.  Try and add a meeting with invalid staffid. | In any case the trigger should block the insert, the error message displayed should correspond to the reason for failure. | As expected |
| meetings\_BEFORE\_update | Try and update a meeting to be only a few minutes after its original time. | The meeting should update successfully. | As expected |
| meetings\_BEFORE\_DELETE | Delete a meeting with a meeting type of lesson, interview or test. | The meeting should be successfully deleted along with the corresponding details in the lesson/interview/test table. | As expected |
| tests\_BEFORE\_INSERT and  tests\_BEFORE\_UPDATE triggers | Change a test type to something other than driving and writing. | The trigger should block the change and send an error message informing the user that the test type is invalid. | As expected |

# 8 Referencing

## 8.1 Sources

Dev.mysql.com. (2017). *MySQL :: MySQL 5.5 Reference Manual :: 12.7 Date and Time Functions*. [online] Available at: https://dev.mysql.com/doc/refman/5.5/en/date-and-time-functions.html [Accessed 17 Dec. 2017].

MySQL?, H. (2017). *How to get an age from a D.O.B field in MySQL?*. [online] Stackoverflow.com. Available at: http://stackoverflow.com/questions/2533890/how-to-get-an-age-from-a-d-o-b-field-in-mysql [Accessed 21 Dec. 2016].

MySQL, B., Procedures, S. and Tutorial, B. (2017). *MySQL Temporary Table | Create, Use and Drop MySQL Temporary Tables*. [online] MySQL Tutorial. Available at: http://www.mysqltutorial.org/mysql-temporary-table/ [Accessed 22 Dec. 2016].

# 9. Appendix

## 9.1 Table Screenshots

### Cars

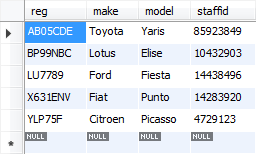


Figure 53 cars table

### Clients

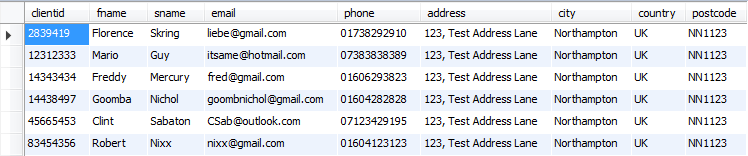


Figure 54 client table

### Faults



Figure 55 faults table

### Interviews

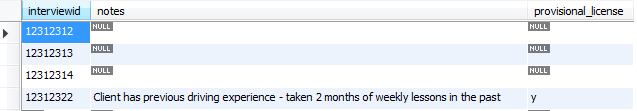


Figure 56 interviews table

### Lessons

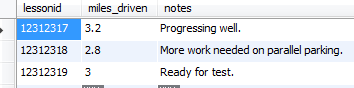


Figure 57 lessons table

### Meetings



Figure 58 meetings table

### office\_staff

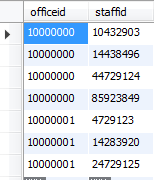


Figure 59 office\_staff table

### Offices

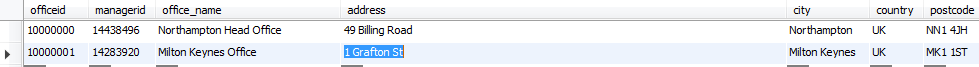


Figure 60 offices table

### Staff

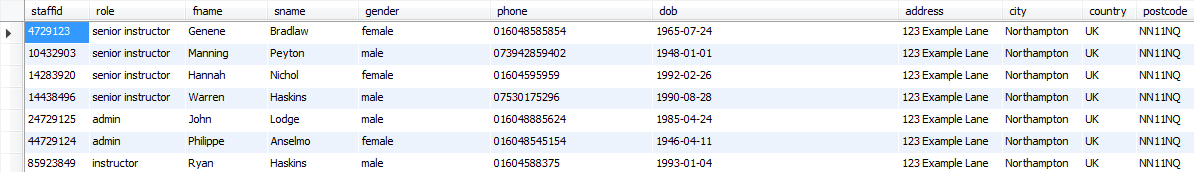


Figure 61 staff table

### tests

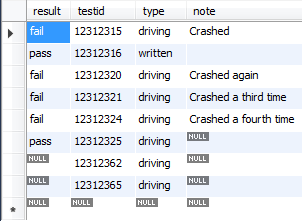


Figure 62 tests table

## 9.2 Database Creation Script – DDL Commands

-- MySQL Script generated by MySQL Workbench

-- 01/04/17 15:49:13

-- Model: New Model Version: 1.0

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='TRADITIONAL,ALLOW\_INVALID\_DATES';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema db14438496

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema db14438496

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `db14438496` DEFAULT CHARACTER SET latin1 ;

USE `db14438496` ;

-- -----------------------------------------------------

-- Table `db14438496`.`staff`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`staff` (

`staffid` INT(11) NOT NULL AUTO\_INCREMENT,

`role` VARCHAR(45) NOT NULL,

`fname` VARCHAR(45) NOT NULL,

`sname` VARCHAR(45) NOT NULL,

`gender` VARCHAR(45) NOT NULL,

`phone` VARCHAR(15) NOT NULL,

`dob` DATE NOT NULL,

`address` VARCHAR(255) NULL DEFAULT NULL,

`city` VARCHAR(45) NULL DEFAULT NULL,

`country` VARCHAR(45) NULL DEFAULT 'UK',

`postcode` VARCHAR(10) NULL DEFAULT NULL,

PRIMARY KEY (`staffid`))

ENGINE = InnoDB

AUTO\_INCREMENT = 85923865

DEFAULT CHARACTER SET = latin1;

-- -----------------------------------------------------

-- Table `db14438496`.`cars`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`cars` (

`reg` VARCHAR(10) NOT NULL,

`make` VARCHAR(50) NULL DEFAULT NULL,

`model` VARCHAR(50) NULL DEFAULT NULL,

`staffid` INT(11) NULL DEFAULT NULL,

PRIMARY KEY (`reg`),

INDEX `staffid\_idx` (`staffid` ASC),

CONSTRAINT `staffid`

FOREIGN KEY (`staffid`)

REFERENCES `db14438496`.`staff` (`staffid`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1;

-- -----------------------------------------------------

-- Table `db14438496`.`clients`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`clients` (

`clientid` INT(11) NOT NULL AUTO\_INCREMENT,

`fname` VARCHAR(45) NULL DEFAULT NULL,

`sname` VARCHAR(45) NULL DEFAULT NULL,

`email` VARCHAR(45) NULL DEFAULT NULL,

`phone` VARCHAR(45) NULL DEFAULT NULL,

`address` VARCHAR(255) NULL DEFAULT NULL,

`city` VARCHAR(45) NULL DEFAULT NULL,

`country` VARCHAR(45) NULL DEFAULT 'UK',

`postcode` VARCHAR(10) NULL DEFAULT NULL,

PRIMARY KEY (`clientid`))

ENGINE = InnoDB

AUTO\_INCREMENT = 83454357

DEFAULT CHARACTER SET = latin1;

-- -----------------------------------------------------

-- Table `db14438496`.`faults`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`faults` (

`faultid` INT(11) NOT NULL AUTO\_INCREMENT,

`reg` VARCHAR(10) NOT NULL,

`description` VARCHAR(1000) NOT NULL,

PRIMARY KEY (`faultid`),

INDEX `fk\_flaws\_cars1` (`reg` ASC),

CONSTRAINT `fk\_flaws\_cars1`

FOREIGN KEY (`reg`)

REFERENCES `db14438496`.`cars` (`reg`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB

AUTO\_INCREMENT = 6

DEFAULT CHARACTER SET = latin1;

-- -----------------------------------------------------

-- Table `db14438496`.`meetings`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`meetings` (

`meetingid` INT(11) NOT NULL AUTO\_INCREMENT,

`datetime` DATETIME NOT NULL,

`clientid` INT(11) NOT NULL,

`staffid` INT(11) NOT NULL,

`type` VARCHAR(45) NOT NULL,

PRIMARY KEY (`meetingid`),

INDEX `fk\_meeting\_clients1\_idx` (`clientid` ASC),

INDEX `fk\_meeting\_staff1\_idx` (`staffid` ASC),

CONSTRAINT `fk\_meeting\_clients1`

FOREIGN KEY (`clientid`)

REFERENCES `db14438496`.`clients` (`clientid`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_meeting\_staff1`

FOREIGN KEY (`staffid`)

REFERENCES `db14438496`.`staff` (`staffid`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB

AUTO\_INCREMENT = 12312362

DEFAULT CHARACTER SET = latin1;

-- -----------------------------------------------------

-- Table `db14438496`.`interviews`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`interviews` (

`interviewid` INT(11) NOT NULL,

`notes` VARCHAR(500) NULL DEFAULT NULL,

`provisional\_license` VARCHAR(1) NULL DEFAULT NULL,

PRIMARY KEY (`interviewid`),

INDEX `fk\_interviews\_meeting1\_idx` (`interviewid` ASC),

CONSTRAINT `fk\_interviews\_meeting1`

FOREIGN KEY (`interviewid`)

REFERENCES `db14438496`.`meetings` (`meetingid`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1;

-- -----------------------------------------------------

-- Table `db14438496`.`lessons`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`lessons` (

`lessonid` INT(11) NOT NULL,

`miles\_driven` DOUBLE NULL DEFAULT NULL,

`notes` VARCHAR(500) NULL DEFAULT NULL,

PRIMARY KEY (`lessonid`),

INDEX `fk\_lessons\_meeting1\_idx` (`lessonid` ASC),

CONSTRAINT `fk\_lessons\_meeting1`

FOREIGN KEY (`lessonid`)

REFERENCES `db14438496`.`meetings` (`meetingid`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1;

-- -----------------------------------------------------

-- Table `db14438496`.`offices`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`offices` (

`officeid` INT(11) NOT NULL AUTO\_INCREMENT,

`managerid` INT(11) NOT NULL,

`office\_name` VARCHAR(45) NULL DEFAULT NULL,

`address` VARCHAR(255) NULL DEFAULT NULL,

`city` VARCHAR(45) NULL DEFAULT NULL,

`country` VARCHAR(45) NULL DEFAULT 'UK',

`postcode` VARCHAR(10) NULL DEFAULT NULL,

PRIMARY KEY (`officeid`),

INDEX `fk\_office\_staff1\_idx` (`managerid` ASC),

CONSTRAINT `fk\_office\_staff1`

FOREIGN KEY (`managerid`)

REFERENCES `db14438496`.`staff` (`staffid`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB

AUTO\_INCREMENT = 10000002

DEFAULT CHARACTER SET = latin1;

-- -----------------------------------------------------

-- Table `db14438496`.`office\_staff`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`office\_staff` (

`officeid` INT(11) NOT NULL,

`staffid` INT(11) NOT NULL,

PRIMARY KEY (`officeid`, `staffid`),

INDEX `fk\_office\_staff\_offices2\_idx` (`officeid` ASC),

INDEX `fk\_office\_staff\_staff2\_idx` (`staffid` ASC),

CONSTRAINT `fk\_office\_staff\_offices2`

FOREIGN KEY (`officeid`)

REFERENCES `db14438496`.`offices` (`officeid`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_office\_staff\_staff2`

FOREIGN KEY (`staffid`)

REFERENCES `db14438496`.`staff` (`staffid`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1;

ALTER TABLE `db14438496`.`office\_staff`

ADD UNIQUE INDEX `staffid\_UNIQUE` (`staffid` ASC);

-- -----------------------------------------------------

-- Table `db14438496`.`tests`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`tests` (

`result` VARCHAR(4) NULL DEFAULT NULL,

`testid` INT(11) NOT NULL,

`type` VARCHAR(45) NOT NULL,

`note` VARCHAR(500) NULL DEFAULT NULL,

PRIMARY KEY (`testid`),

INDEX `fk\_tests\_meeting1\_idx` (`testid` ASC),

CONSTRAINT `fk\_tests\_meeting1`

FOREIGN KEY (`testid`)

REFERENCES `db14438496`.`meetings` (`meetingid`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1;

USE `db14438496` ;

-- -----------------------------------------------------

-- Placeholder table for view `db14438496`.`avg\_miles`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`avg\_miles` (`AVG(miles\_driven)` INT);

-- -----------------------------------------------------

-- Placeholder table for view `db14438496`.`faultless\_cars`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`faultless\_cars` (`reg` INT, `make` INT, `model` INT);

-- -----------------------------------------------------

-- Placeholder table for view `db14438496`.`instructors\_over\_55`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`instructors\_over\_55` (`staffid` INT, `fname` INT, `sname` INT, `age` INT, `phone` INT);

-- -----------------------------------------------------

-- Placeholder table for view `db14438496`.`office\_managers`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`office\_managers` (`officeid` INT, `office\_name` INT, `fname` INT, `sname` INT, `phone` INT);

-- -----------------------------------------------------

-- Placeholder table for view `db14438496`.`total\_staff\_per\_office`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `db14438496`.`total\_staff\_per\_office` (`officeid` INT, `office\_name` INT, `COUNT(s.staffid)` INT);

-- -----------------------------------------------------

-- procedure check\_client\_availability

-- -----------------------------------------------------

DELIMITER $$

USE `db14438496`$$

CREATE DEFINER=`s14438496`@`%` PROCEDURE `check\_client\_availability`(

IN id INT,

IN date\_time DATETIME,

IN new\_type VARCHAR(20),

OUT available BOOLEAN)

BEGIN

DECLARE done BOOLEAN DEFAULT 0;

DECLARE time\_holder DATETIME;

DECLARE meeting\_type VARCHAR(20);

DECLARE times CURSOR FOR

SELECT m.datetime, m.type

FROM meetings m

WHERE clientid = id;

DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET

done=1;

SET available = true;

OPEN times;

the\_loop: LOOP

IF done THEN LEAVE the\_loop; END IF;

FETCH times INTO time\_holder, meeting\_type;

IF date\_diff(date\_time, time\_holder, meeting\_type, new\_type) = false THEN

SET available = false;

END IF;

END LOOP;

CLOSE times;

END$$

DELIMITER ;

-- -----------------------------------------------------

-- procedure check\_client\_availability\_update

-- -----------------------------------------------------

DELIMITER $$

USE `db14438496`$$

CREATE DEFINER=`s14438496`@`%` PROCEDURE `check\_client\_availability\_update`(

IN id INT,

IN date\_time DATETIME,

IN new\_type VARCHAR(20),

IN meeting\_id INT,

OUT available BOOLEAN)

BEGIN

DECLARE done BOOLEAN DEFAULT 0;

DECLARE time\_holder DATETIME;

DECLARE meeting\_type VARCHAR(20);

DECLARE times CURSOR FOR

SELECT m.datetime, m.type

FROM meetings m

WHERE clientid = id

AND m.meetingid != meeting\_id;

DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET

done=1;

SET available = true;

OPEN times;

the\_loop: LOOP

IF done THEN LEAVE the\_loop; END IF;

FETCH times INTO time\_holder, meeting\_type;

IF date\_diff(date\_time, time\_holder, meeting\_type, new\_type) = false THEN

SET available = false;

END IF;

END LOOP;

CLOSE times;

END$$

DELIMITER ;

-- -----------------------------------------------------

-- procedure check\_staff\_availability

-- -----------------------------------------------------

DELIMITER $$

USE `db14438496`$$

CREATE DEFINER=`s14438496`@`%` PROCEDURE `check\_staff\_availability`(

IN id INT,

IN date\_time DATETIME,

IN new\_type VARCHAR(20),

OUT available BOOLEAN)

BEGIN

DECLARE done BOOLEAN DEFAULT 0;

DECLARE time\_holder DATETIME;

DECLARE meeting\_type VARCHAR(20);

DECLARE times CURSOR FOR

SELECT m.datetime, m.type

FROM meetings m

WHERE staffid = id;

DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET

done=1;

SET available = true;

OPEN times;

the\_loop: LOOP

IF done THEN LEAVE the\_loop; END IF;

FETCH times INTO time\_holder, meeting\_type;

IF date\_diff(date\_time, time\_holder, meeting\_type, new\_type) = false THEN

SET available = false;

END IF;

END LOOP;

CLOSE times;

END$$

DELIMITER ;

-- -----------------------------------------------------

-- procedure check\_staff\_availability\_update

-- -----------------------------------------------------

DELIMITER $$

USE `db14438496`$$

CREATE DEFINER=`s14438496`@`%` PROCEDURE `check\_staff\_availability\_update`(

IN staff\_id INT,

IN date\_time DATETIME,

IN new\_type VARCHAR(20),

IN meeting\_id INT,

OUT available BOOLEAN)

BEGIN

DECLARE done BOOLEAN DEFAULT 0;

DECLARE time\_holder DATETIME;

DECLARE meeting\_type VARCHAR(20);

DECLARE times CURSOR FOR

SELECT m.datetime, m.type

FROM meetings m

WHERE staffid = staff\_id

AND m.meetingid != meeting\_id;

DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET

done=1;

SET available = true;

OPEN times;

the\_loop: LOOP

IF done THEN LEAVE the\_loop; END IF;

FETCH times INTO time\_holder, meeting\_type;

IF date\_diff(date\_time, time\_holder, meeting\_type, new\_type) = false THEN

SET available = false;

END IF;

END LOOP;

CLOSE times;

END$$

DELIMITER ;

-- -----------------------------------------------------

-- function date\_diff

-- -----------------------------------------------------

DELIMITER $$

USE `db14438496`$$

CREATE DEFINER=`s14438496`@`%` FUNCTION `date\_diff`(new\_time DATETIME,old\_time DATETIME, old\_type VARCHAR(20), new\_type VARCHAR(20)) RETURNS tinyint(1)

BEGIN

DECLARE difference INT;

DECLARE available BOOLEAN;

DECLARE pre\_threshold INT;

DECLARE post\_threshold INT;

SET difference = TIMESTAMPDIFF(MINUTE,new\_time,old\_time);

SET available = true;

IF new\_type = 'pre test meeting' OR new\_type = 'post test meeting' THEN

SET pre\_threshold = 10;

ELSE

SET pre\_threshold = 60;

END IF;

IF old\_type = 'pre test meeting' OR old\_type = 'post test meeting' THEN

SET post\_threshold = -10;

ELSE

SET post\_threshold = -60;

END IF;

IF difference < pre\_threshold AND difference > post\_threshold THEN

SET available = false;

END IF;

RETURN available;

END$$

DELIMITER ;

-- -----------------------------------------------------

-- procedure females\_in\_office

-- -----------------------------------------------------

DELIMITER $$

USE `db14438496`$$

CREATE DEFINER=`s14438496`@`%` PROCEDURE `females\_in\_office`(

IN officeid INT)

BEGIN

SELECT s.fname, s.sname, s.gender

FROM staff s

JOIN office\_staff os

ON s.staffid = os.staffid

WHERE os.officeid = officeid

AND s.gender = 'female';

END$$

DELIMITER ;

-- -----------------------------------------------------

-- procedure find\_meeting\_id

-- -----------------------------------------------------

DELIMITER $$

USE `db14438496`$$

CREATE DEFINER=`s14438496`@`%` PROCEDURE `find\_meeting\_id`(

IN date\_time DATETIME,

IN staff\_id INT,

OUT found\_id INT)

BEGIN

DECLARE fetch\_id CURSOR FOR

SELECT meetingid FROM meetings WHERE datetime = date\_time AND staffid = staff\_id;

OPEN fetch\_id;

FETCH fetch\_id INTO found\_id;

CLOSE fetch\_id;

END$$

DELIMITER ;

-- -----------------------------------------------------

-- procedure interviews\_conducted

-- -----------------------------------------------------

DELIMITER $$

USE `db14438496`$$

CREATE DEFINER=`s14438496`@`%` PROCEDURE `interviews\_conducted`(

IN staffid INT)

BEGIN

SELECT s.staffid, s.fname, s.sname, m.datetime, i.interviewid, i.notes, i.provisional\_license

FROM staff s

JOIN meetings m

ON s.staffid = m.staffid

JOIN interviews i

ON m.meetingid = i.interviewid

WHERE s.staffid = staffid

AND m.datetime < CURRENT\_DATE();

END$$

DELIMITER ;

-- -----------------------------------------------------

-- procedure passes\_in\_month

-- -----------------------------------------------------

DELIMITER $$

USE `db14438496`$$

CREATE DEFINER=`s14438496`@`%` PROCEDURE `passes\_in\_month`(

IN mnth INT,

IN yr INT)

BEGIN

SELECT c.fname, c.sname

FROM clients c

JOIN meetings m

ON m.clientid = c.clientid

JOIN tests t

ON t.testid = m.meetingid

WHERE MONTH(m.datetime) = mnth

AND YEAR(m.datetime) = yr

AND t.result = 'pass'

AND t.type = 'driving';

END$$

DELIMITER ;

-- -----------------------------------------------------

-- procedure schedule\_test

-- -----------------------------------------------------

DELIMITER $$

USE `db14438496`$$

CREATE DEFINER=`s14438496`@`%` PROCEDURE `schedule\_test`(

IN client\_id INT,

IN tester\_id INT,

IN instructor\_id INT,

IN date\_time DATETIME,

IN test\_type VARCHAR(10)

)

BEGIN

DECLARE id\_holder INT DEFAULT 0;

INSERT INTO meetings(datetime, clientid, staffid, type)

VALUES (date\_time, client\_id, tester\_id, 'test');

CALL find\_meeting\_id(date\_time, tester\_id, id\_holder);

IF id\_holder != 0 THEN

INSERT INTO tests(testid, type)

VALUES (id\_holder, test\_type);

INSERT INTO meetings(datetime, staffid, clientid, type)

VALUES (ADDTIME(date\_time, '01:00:00'),instructor\_id, client\_id, 'post test meeting');

INSERT INTO meetings(datetime, staffid, clientid, type)

VALUES (DATE\_SUB(date\_time,INTERVAL 10 MINUTE), instructor\_id, client\_id, 'pre test meeting');

END IF;

END$$

DELIMITER ;

-- -----------------------------------------------------

-- procedure staff\_meetings\_in\_next\_week

-- -----------------------------------------------------

DELIMITER $$

USE `db14438496`$$

CREATE DEFINER=`s14438496`@`%` PROCEDURE `staff\_meetings\_in\_next\_week`(

IN staff\_id INT

)

BEGIN

SELECT s.staffid, s.fname, s.sname, m.datetime, m.type

FROM staff s

JOIN meetings m

ON s.staffid = m.staffid

WHERE TIMESTAMPDIFF(DAY,NOW(), m.datetime) BETWEEN 7 AND 14

AND s.staffid = staff\_id;

END$$

DELIMITER ;

-- -----------------------------------------------------

-- procedure triple\_fail\_check

-- -----------------------------------------------------

DELIMITER $$

USE `db14438496`$$

CREATE DEFINER=`s14438496`@`%` PROCEDURE `triple\_fail\_check`()

BEGIN

DECLARE done BOOLEAN DEFAULT 0;

DECLARE newid INT DEFAULT 1;

DECLARE cid INT;

DECLARE cfname VARCHAR(50);

DECLARE csname VARCHAR(50);

DECLARE cfails INT;

DECLARE fails CURSOR FOR

SELECT c.clientid, c.fname, c.sname, COUNT(\*) as fails

FROM clients c

JOIN meetings m

ON c.clientid = m.clientid

JOIN tests t

ON t.testid = m.meetingid

WHERE t.result = 'fail'

AND t.type = 'driving'

AND c.clientid NOT IN(

SELECT c.clientid

FROM clients c

JOIN meetings m

ON c.clientid = m.clientid

JOIN tests t

ON t.testid = m.meetingid

WHERE t.result = 'pass'

AND t.type = 'driving')

GROUP BY c.fname;

DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET

done=1;

CREATE TEMPORARY TABLE `temp` (

`id` INT NOT NULL,

`cid` INT NOT NULL,

`firstname` VARCHAR(50) NULL,

`surname` VARCHAR(50) NULL,

`fails` INT NULL,

PRIMARY KEY (`id`));

OPEN fails;

the\_loop: LOOP

IF done THEN LEAVE the\_loop; END IF;

FETCH fails INTO cid, cfname, csname, cfails;

IF cfails > 3 THEN INSERT INTO temp VALUES (newid, cid, cfname, csname, cfails); end if;

SET newid = newid + 1;

END LOOP;

CLOSE fails;

SELECT \* FROM temp

GROUP BY cid;

DROP TABLE temp;

END$$

DELIMITER ;

-- -----------------------------------------------------

-- View `db14438496`.`avg\_miles`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `db14438496`.`avg\_miles`;

USE `db14438496`;

CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=`s14438496`@`%` SQL SECURITY DEFINER VIEW `db14438496`.`avg\_miles` AS select avg(`db14438496`.`lessons`.`miles\_driven`) AS `AVG(miles\_driven)` from `db14438496`.`lessons`;

-- -----------------------------------------------------

-- View `db14438496`.`faultless\_cars`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `db14438496`.`faultless\_cars`;

USE `db14438496`;

CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=`s14438496`@`%` SQL SECURITY DEFINER VIEW `db14438496`.`faultless\_cars` AS select `db14438496`.`cars`.`reg` AS `reg`,`db14438496`.`cars`.`make` AS `make`,`db14438496`.`cars`.`model` AS `model` from `db14438496`.`cars` where (not(`db14438496`.`cars`.`reg` in (select `db14438496`.`faults`.`reg` from `db14438496`.`faults`)));

-- -----------------------------------------------------

-- View `db14438496`.`instructors\_over\_55`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `db14438496`.`instructors\_over\_55`;

USE `db14438496`;

CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=`s14438496`@`%` SQL SECURITY DEFINER VIEW `db14438496`.`instructors\_over\_55` AS select `db14438496`.`staff`.`staffid` AS `staffid`,`db14438496`.`staff`.`fname` AS `fname`,`db14438496`.`staff`.`sname` AS `sname`,((year(curdate()) - year(`db14438496`.`staff`.`dob`)) - (date\_format(curdate(),'00-%m-%d') < date\_format(`db14438496`.`staff`.`dob`,'00-%m-%d'))) AS `age`,`db14438496`.`staff`.`phone` AS `phone` from `db14438496`.`staff` where ((`db14438496`.`staff`.`role` like '%instructor') and (((year(curdate()) - year(`db14438496`.`staff`.`dob`)) - (date\_format(curdate(),'00-%m-%d') < date\_format(`db14438496`.`staff`.`dob`,'00-%m-%d'))) > 55));

-- -----------------------------------------------------

-- View `db14438496`.`office\_managers`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `db14438496`.`office\_managers`;

USE `db14438496`;

CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=`s14438496`@`%` SQL SECURITY DEFINER VIEW `db14438496`.`office\_managers` AS select `o`.`officeid` AS `officeid`,`o`.`office\_name` AS `office\_name`,`s`.`fname` AS `fname`,`s`.`sname` AS `sname`,`s`.`phone` AS `phone` from (`db14438496`.`staff` `s` join `db14438496`.`offices` `o` on((`s`.`staffid` = `o`.`managerid`))) where (`s`.`staffid` = `o`.`managerid`);

-- -----------------------------------------------------

-- View `db14438496`.`total\_staff\_per\_office`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `db14438496`.`total\_staff\_per\_office`;

USE `db14438496`;

CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=`s14438496`@`%` SQL SECURITY DEFINER VIEW `db14438496`.`total\_staff\_per\_office` AS select `o`.`officeid` AS `officeid`,`o`.`office\_name` AS `office\_name`,count(`s`.`staffid`) AS `COUNT(s.staffid)` from ((`db14438496`.`offices` `o` join `db14438496`.`office\_staff` `os` on((`o`.`officeid` = `os`.`officeid`))) join `db14438496`.`staff` `s` on((`os`.`staffid` = `s`.`staffid`))) group by `o`.`officeid`;

USE `db14438496`;

DELIMITER $$

USE `db14438496`$$

CREATE

DEFINER=`s14438496`@`%`

TRIGGER `db14438496`.`faults\_BEFORE\_INSERT`

BEFORE INSERT ON `db14438496`.`faults`

FOR EACH ROW

BEGIN

IF new.reg NOT IN (SELECT reg FROM cars) THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Not a recognised car. Add the car to the list of registered cars first.';

END IF;

END$$

USE `db14438496`$$

CREATE

DEFINER=`s14438496`@`%`

TRIGGER `db14438496`.`faults\_BEFORE\_UPDATE`

BEFORE UPDATE ON `db14438496`.`faults`

FOR EACH ROW

BEGIN

IF new.reg NOT IN (SELECT reg FROM cars) THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Not a recognised car. Add the car to the list of registered cars first.';

END IF;

END$$

USE `db14438496`$$

CREATE

DEFINER=`s14438496`@`%`

TRIGGER `db14438496`.`meetings\_BEFORE\_INSERT`

BEFORE INSERT ON `db14438496`.`meetings`

FOR EACH ROW

BEGIN

DECLARE available BOOLEAN;

CALL check\_staff\_availability(new.staffid, new.datetime, new.type, available);

IF available = false THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Instructor is busy at that time.';

END IF;

CALL check\_client\_availability(new.clientid, new.datetime, new.type, available);

IF available = false THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Client is already booked in at that time.';

END IF;

IF new.type != 'lesson' AND new.type != 'test' AND new.type != 'interview' AND new.type != 'pre test meeting' AND new.type != 'post test meeting' THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Invalid meeting type';

END IF;

IF new.staffid NOT IN (SELECT staffid FROM staff) THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Not a recognised staff ID';

END IF;

IF new.clientid NOT IN (SELECT clientid FROM clients) THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Not a recognised client ID';

END IF;

END$$

USE `db14438496`$$

CREATE

DEFINER=`s14438496`@`%`

TRIGGER `db14438496`.`meetings\_BEFORE\_UPDATE`

BEFORE UPDATE ON `db14438496`.`meetings`

FOR EACH ROW

BEGIN

DECLARE available BOOLEAN;

IF TIMESTAMPDIFF(MINUTE, NEW.datetime, OLD.datetime) != 0 THEN

CALL check\_staff\_availability\_update(new.staffid, new.datetime, new.type, old.meetingid, available);

IF available = false THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Instructor is busy at that time.';

END IF;

CALL check\_client\_availability\_update(new.clientid, new.datetime, new.type, old.meetingid, available);

IF available = false THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Client is already booked in at that time.';

END IF;

END IF;

IF new.type != 'lesson' AND new.type != 'test' AND new.type != 'interview' AND new.type != 'pre-test meeting' AND new.type != 'post-test meeting' THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Invalid meeting type';

END IF;

IF new.staffid NOT IN (SELECT staffid FROM staff) THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Not a recognised staff ID';

END IF;

IF new.clientid NOT IN (SELECT clientid FROM clients) THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Not a recognised client ID';

END IF;

END$$

USE `db14438496`$$

CREATE

DEFINER=`s14438496`@`%`

TRIGGER `db14438496`.`meetings\_BEFORE\_DELETE`

BEFORE DELETE ON `db14438496`.`meetings`

FOR EACH ROW

BEGIN

IF old.type = 'lesson' THEN

DELETE FROM lessons WHERE lessonid = old.meetingid;

END IF;

IF old.type = 'interview' THEN

DELETE FROM interviews WHERE interviewid = old.meetingid;

END IF;

IF old.type = 'test' THEN

DELETE FROM tests WHERE testid = old.meetingid;

END IF;

END$$

USE `db14438496`$$

CREATE

DEFINER=`s14438496`@`%`

TRIGGER `db14438496`.`interviews\_BEFORE\_INSERT`

BEFORE INSERT ON `db14438496`.`interviews`

FOR EACH ROW

BEGIN

IF new.provisional\_license != 'y' AND new.provisional\_license != 'n' THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Invalid input for provisional license. Valid inputs are "y" or "n".';

END IF;

IF new.interviewid NOT IN (SELECT meetingid FROM meetings) THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Not a recognised meeting.';

END IF;

END$$

USE `db14438496`$$

CREATE

DEFINER=`s14438496`@`%`

TRIGGER `db14438496`.`interviews\_BEFORE\_UPDATE`

BEFORE UPDATE ON `db14438496`.`interviews`

FOR EACH ROW

BEGIN

IF new.provisional\_license != 'y' AND new.provisional\_license != 'n' THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Invalid input for provisional license. Valid inputs are "y" or "n".';

END IF;

IF new.interviewid NOT IN (SELECT meetingid FROM meetings) THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Not a recognised meeting.';

END IF;

END$$

USE `db14438496`$$

CREATE

DEFINER=`s14438496`@`%`

TRIGGER `db14438496`.`lessons\_BEFORE\_INSERT`

BEFORE INSERT ON `db14438496`.`lessons`

FOR EACH ROW

BEGIN

IF new.lessonid NOT IN (SELECT meetingid FROM meetings) THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Not a recognised meeting.';

END IF;

END$$

USE `db14438496`$$

CREATE

DEFINER=`s14438496`@`%`

TRIGGER `db14438496`.`lessons\_BEFORE\_UPDATE`

BEFORE UPDATE ON `db14438496`.`lessons`

FOR EACH ROW

BEGIN

IF new.lessonid NOT IN (SELECT meetingid FROM meetings) THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Not a recognised meeting.';

END IF;

END$$

USE `db14438496`$$

CREATE

DEFINER=`s14438496`@`%`

TRIGGER `db14438496`.`tests\_BEFORE\_INSERT`

BEFORE INSERT ON `db14438496`.`tests`

FOR EACH ROW

BEGIN

IF new.type != 'driving' AND new.type != 'written' THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Invalid test type';

END IF;

END$$

USE `db14438496`$$

CREATE

DEFINER=`s14438496`@`%`

TRIGGER `db14438496`.`tests\_BEFORE\_UPDATE`

BEFORE UPDATE ON `db14438496`.`tests`

FOR EACH ROW

BEGIN

IF new.type != 'driving' AND new.type != 'written' THEN

SIGNAL SQLSTATE '45000' SET message\_text = 'Invalid test type';

END IF;

END$$

DELIMITER ;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

## 9.3 Test Data Script

INSERT INTO staff (staffid, fname, sname, role, gender, phone, dob, address, city, country, postcode)

VALUES (14438496, 'Warren', 'Haskins', 'senior instructor', 'male', '07530175296', 19900828,'123 Example Lane', 'Northampton', 'UK', 'NN11NQ');

INSERT INTO staff (staffid, fname, sname, role, gender, phone, dob, address, city, country, postcode)

VALUES (14283920, 'Hannah', 'Nichol', 'senior instructor', 'female', '01604595959', 19920226,'123 Example Lane', 'Northampton', 'UK', 'NN11NQ');

INSERT INTO staff (staffid, fname, sname, role, gender, phone, dob, address, city, country, postcode)

VALUES (85923849, 'Ryan', 'Haskins', 'instructor', 'male', '01604588375', 19930104,'123 Example Lane', 'Northampton', 'UK', 'NN11NQ');

INSERT INTO staff (staffid, fname, sname, role, gender, phone, dob, address, city, country, postcode)

VALUES (10432903, 'Manning', 'Peyton', 'senior instructor', 'male', '073942859402', 19480101,'123 Example Lane', 'Northampton', 'UK', 'NN11NQ');

INSERT INTO staff (staffid, fname, sname, role, gender, phone, dob, address, city, country, postcode)

VALUES (04729123, 'Genene', 'Bradlaw', 'senior instructor', 'female', '016048585854', 19650724,'123 Example Lane', 'Northampton', 'UK', 'NN11NQ');

INSERT INTO staff (staffid, fname, sname, role, gender, phone, dob, address, city, country, postcode)

VALUES (44729124, 'Philippe', 'Anselmo', 'admin', 'female', '016048545154', 19460411,'123 Example Lane', 'Northampton', 'UK', 'NN11NQ');

INSERT INTO staff (staffid, fname, sname, role, gender, phone, dob, address, city, country, postcode)

VALUES (24729125, 'John', 'Lodge', 'admin', 'male', '016048885624', 19850424,'123 Example Lane', 'Northampton', 'UK', 'NN11NQ');

INSERT INTO clients (clientid, fname, sname, email, phone, address, city, country, postcode)

VALUES (14438497, 'Goomba', 'Nichol', 'goombnichol@gmail.com', '01604282828', '123, Test Address Lane', 'Northampton', 'UK', 'NN1123');

INSERT INTO clients (clientid, fname, sname, email, phone, address, city, country, postcode)

VALUES (14343434, 'Freddy', 'Mercury', 'fred@gmail.com', '01606293823', '123, Test Address Lane', 'Northampton', 'UK', 'NN1123');

INSERT INTO clients (clientid, fname, sname, email, phone, address, city, country, postcode)

VALUES (12312333, 'Mario', 'Guy', 'itsame@hotmail.com', '07383838389', '123, Test Address Lane', 'Northampton', 'UK', 'NN1123');

INSERT INTO clients (clientid, fname, sname, email, phone, address, city, country, postcode)

VALUES (45665453, 'Clint', 'Sabaton', 'CSab@outlook.com', '07123429195', '123, Test Address Lane', 'Northampton', 'UK', 'NN1123');

INSERT INTO clients (clientid, fname, sname, email, phone, address, city, country, postcode)

VALUES (83454356, 'Robert', 'Nixx', 'nixx@gmail.com', '01604123123', '123, Test Address Lane', 'Northampton', 'UK', 'NN1123');

INSERT INTO clients (clientid, fname, sname, email, phone, address, city, country, postcode)

VALUES (02839419, 'Florence', 'Skring', 'liebe@gmail.com', '01738292910', '123, Test Address Lane', 'Northampton', 'UK', 'NN1123');

INSERT INTO cars(reg, make, model, staffid)

VALUES ('X631ENV', 'Fiat', 'Punto', 14283920);

INSERT INTO cars(reg, make, model, staffid)

VALUES ('AB05CDE', 'Toyota', 'Yaris', 85923849);

INSERT INTO cars(reg, make, model, staffid)

VALUES ('LU7789', 'Ford', 'Fiesta', 14438496);

INSERT INTO cars(reg, make, model, staffid)

VALUES ('BP99NBC', 'Lotus', 'Elise', 10432903);

INSERT INTO cars(reg, make, model, staffid)

VALUES ('YLP75F', 'Citroen', 'Picasso', 04729123);

INSERT INTO offices(officeid, office\_name, address, city, country, postcode, managerid)

VALUES (10000000, 'Northampton Head Office','49 Billing Road', 'Northampton','UK', 'NN1 4JH', 14438496);

INSERT INTO offices(officeid, office\_name, address, city, country, postcode, managerid)

VALUES (10000001, 'Milton Keynes Office','1 Grafton St', 'Milton Keynes', 'UK', 'MK1 1ST', 14283920);

INSERT INTO office\_staff(officeid, staffid)

VALUES(10000000, 14438496);

INSERT INTO office\_staff(officeid, staffid)

VALUES(10000001, 14283920);

INSERT INTO office\_staff(officeid, staffid)

VALUES(10000000, 85923849);

INSERT INTO office\_staff(officeid, staffid)

VALUES(10000000, 10432903);

INSERT INTO office\_staff(officeid, staffid)

VALUES(10000001, 04729123);

INSERT INTO office\_staff(officeid, staffid)

VALUES(10000000, 44729124);

INSERT INTO office\_staff(officeid, staffid)

VALUES(10000001, 24729125);

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312312, '2017-02-10 13:00:00', 14438497, 14438496, 'interview');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312313, '2017-02-15 11:30:00', 12312333, 85923849, 'interview');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312314, '2017-01-15 18:15:00', 83454356, 04729123, 'interview');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312322, '2016-11-18 13:15:00', 14438497, 14438496, 'interview');

INSERT INTO interviews(interviewid)

VALUES (12312312);

INSERT INTO interviews(interviewid)

VALUES (12312313);

INSERT INTO interviews(interviewid)

VALUES (12312314);

INSERT INTO interviews(interviewid, notes, provisional\_license)

VALUES (12312322, 'Client has previous driving experience - taken 2 months of weekly lessons in the past', 'y');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312315, '2016-12-23 09:30:00', 14343434, 14438496, 'test');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312316, '2017-01-20 17:15:00', 45665453, 10432903, 'test');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312320, '2016-12-21 12:30:00', 14343434, 14438496, 'test');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312321, '2016-12-22 14:30:00', 14343434, 14438496, 'test');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312324, '2017-01-09 14:00:00', 14343434, 14438496, 'test');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312325, '2017-01-09 14:00:00', 45665453, 10432903, 'test');

-- Past tests

INSERT INTO tests(testid, result, type, note)

VALUES (12312315, 'fail', 'driving', 'Crashed');

INSERT INTO tests(testid, result, type, note)

VALUES (12312316, 'pass', 'written', '');

INSERT INTO tests(testid, result, type, note)

VALUES (12312320, 'fail', 'driving', 'Crashed again');

INSERT INTO tests(testid, result, type, note)

VALUES (12312321, 'fail', 'driving', 'Crashed a third time');

INSERT INTO tests(testid, result, type, note)

VALUES (12312324, 'fail', 'driving', 'Crashed a fourth time');

INSERT INTO tests(testid, result, type)

VALUES (12312325, 'pass', 'driving');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312317, '2016-12-05 09:20:00', 14438497, 14438496, 'lesson');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312318, '2016-12-05 11:45:00', 12312333, 85923849, 'lesson');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312319, '2016-11-24 16:00:00', 83454356, 04729123, 'lesson');

INSERT INTO meetings(meetingid, datetime, clientid, staffid, type)

VALUES (12312323, '2017-02-24 16:00:00', 83454356, 04729123, 'lesson');

INSERT INTO lessons(lessonid, miles\_driven, notes)

VALUES (12312317, 3.2, 'Progressing well.');

INSERT INTO lessons(lessonid, miles\_driven, notes)

VALUES (12312318, 2.8, 'More work needed on parallel parking.');

INSERT INTO lessons(lessonid, miles\_driven, notes)

VALUES (12312319, 3, 'Ready for test.');

INSERT INTO lessons(lessonid)

VALUES (12312323);

-- Future tests

call db14438496.schedule\_test(14343434, 14438496, 85923849, '2017-01-22 16:00:00', 'driving');

call db14438496.schedule\_test(14343434, 14438496, 85923849, '2017-01-21 18:00:00', 'written');

INSERT INTO faults(faultid, reg, description)

VALUES (1, 'X631ENV', 'Faulty headgasket');

INSERT INTO faults(faultid, reg, description)

VALUES (2, 'X631ENV', 'Broken taillight');

INSERT INTO faults(faultid, reg, description)

VALUES (3, 'AB05CDE', 'Faulty seatbelt');

INSERT INTO faults(faultid, reg, description)

VALUES (4, 'AB05CDE', 'Worn clutch');

INSERT INTO faults(faultid, reg, description)

VALUES (5, 'AB05CDE', 'Broken windscreen wipers');

## 9.4 CRUD GUI Code

### 9.4.1 header.php

<?php

session\_start();

echo'<!DOCTYPE HTML>

<html>

<head>

<meta charset="UTF-8">

<title>Warren\'s Website</title>

<link rel="stylesheet" type="text/css" href="styles.css" />

</head>

<header>

<h3> Databases 3 - Assignment 1 - Warren Haskins</h3>

</header>';

$server = '194.81.104.22';

$username = 's14438496';

$password = '14438496';

$schema = 'db14438496';

$date = new DateTime();

$pdo = new PDO('mysql:dbname=' . $schema . ';host=' . $server, $username, $password);

function autoload($name){

require strtolower($name) . '.php';

}

spl\_autoload\_register('autoload');

?>

### 9.4.2 index.php

<?php

require 'header.php';

require 'functions.php';

$table = new addTable();

echo '<div class="mainbody">

<h1>Staff</h1>';

if(isset($\_POST['all\_staff'])){ // if All Staff button is pressed

allStaff($pdo, $table);

}

if(isset($\_POST['alter'])){ // if alter a staff button is pressed

amendStaff($pdo, $\_POST['alter']);

}

if(isset($\_POST['delete'])){ // if delete staff button is pressed

deleteStaff($pdo, $\_POST['delete']);

allStaff($pdo, $table);

}

if(isset($\_POST['submit\_changes'])){ // submit button for altering a staff member

submitChanges($pdo);

allStaff($pdo, $table);

}

if(isset($\_POST['new\_staff'])){ // if add staff button is pessed

newStaff($pdo);

}

if(isset($\_POST['submit\_new\_staff'])){ // submit button for adding new staff

submitNewStaff($pdo);

allStaff($pdo, $table);

}

echo'<form action="index.php" method="POST" enctype="multipart/form-data">

<button type="submit" name="all\_staff">List Staff</button><br>

<button type="submit" name="new\_staff">Add Staff</button><br>

</form>';

echo '</div><br>';

require 'footer.php';

?>

### 9.4.3 functions.php

<?php

function allStaff($pdo, $table){

$staff = $pdo->query('SELECT staffid, role, fname, sname, gender, phone, dob, address, city, country, postcode FROM staff;');

if(!$staff){

echo $pdo->errorInfo()[2];

}else{

$tableHeadings = ['Staff ID', 'role', 'Staff Name', 'Gender', 'Phone Number', 'DOB', 'Office', 'Address'];

$table -> setHeadings($tableHeadings);

foreach($staff as $row){

$alter = '<button type="submit" name="alter" value="' . $row['staffid'] . '">Amend</button>';

$delete = '<button type="submit" name="delete" value="' . $row['staffid'] . '">Delete</button>';

$address = $row['address'] . ' ' . $row['city'] . ' ' . $row['country'] . ' ' . $row['postcode'];

$officeQuery = $pdo->query('SELECT o.office\_name FROM offices o JOIN office\_staff os ON o.officeid = os.officeid WHERE os.staffid = "' . $row['staffid'] . '";');

$office = $officeQuery->fetch();

$rowToAdd = [$row['staffid'], $row['role'], $row['fname'] . ' ' . $row['sname'], $row['gender'], $row['phone'], $row['dob'], $office['office\_name'], $address, $alter, $delete];

$table->addRow($rowToAdd);

}

echo '<form action="index.php" method="POST" enctype="multipart/form-data">';

echo $table->getHTML();

echo '</form>';

}

}

function amendStaff($pdo, $staffid){

$query = $pdo->query('SELECT s.role, s.fname, s.sname, s.gender, s.phone, s.dob, s.address, o.officeid, o.office\_name, s.city, s.country, s.postcode

FROM staff s

JOIN office\_staff os

ON s.staffid = os.staffid

JOIN offices o

ON os.officeid = o.officeid

WHERE s.staffid = ' . $staffid);

if(!$query){

echo $pdo->errorInfo()[2];

}else{

$staff = $query->fetch();

$officeQuery = $pdo->query('SELECT officeid, office\_name FROM offices;');

$default = true;

echo'<form action="index.php" method="POST" enctype="multipart/form-data">

<label for="staffid">ID: ' . $staffid .

'<br><label for="role"> Choose role:

<select name="role">

<option value="senior instructor" '; if($staff['role'] == 'senior instructor'){echo 'selected'; $default = false;} echo '>Senior Instructor</option>

<option value="instructor" '; if($staff['role'] == 'instructor'){echo 'selected'; $default = false;} echo '>Instructor</option>

<option value="admin" '; if($staff['role'] == 'admin'){echo 'selected'; $default = false;} echo '>Administrative Staff</option>

<option value="other" '; if($default){echo 'selected';} echo '>Other</option>

</select><br>

<label for="fname">First Name:

<input name= "fname" type="text" value="' . $staff['fname'] . '"><br>

<label for="sname">Surname:

<input name= "sname" type="text" value="' . $staff['sname'] . '"><br>

<label for="gender">Gender:

<input name= "gender" type="text" value="' . $staff['gender'] . '"><br>

<label for="phone">Phone Number:

<input name= "phone" type="text" value="' . $staff['phone'] . '"><br>

<label for="dob">DOB:

<input name= "dob" type="text" value="' . $staff['dob'] . '">

<br><label for="office"> Choose office:

<select name="office">';

foreach($officeQuery as $row){

echo'<option value="' . $row['officeid'] . '" ';

if($row['officeid'] == $staff['officeid']){

echo 'selected';;

}

echo '>' . $row['office\_name'] . '</option>';

}

echo '</select><br>

<label for="address">Address:

<input name= "address" type="text" value="' . $staff['address'] . '"><br>

<label for="city">City:

<input name= "city" type="text" value="' . $staff['city'] . '"><br>

<label for="country">Country:

<input name= "country" type="text" value="' . $staff['country'] . '"><br>

<label for="postcode">Postcode:

<input name= "postcode" type="text" value="' . $staff['postcode'] . '"><br>

<button type="submit" name="submit\_changes" value="' . $staffid . '"> Submit </button><br>

</form>';

}

}

function submitChanges($pdo){

$stmt = $pdo->prepare('UPDATE staff SET role = :role, fname = :fname, sname = :sname, gender = :gender, phone = :phone, dob = :dob,

address = :address, city = :city, country = :country, postcode = :postcode

WHERE staffid = ' . $\_POST['submit\_changes']);

$criteria = [

'role' => $\_POST['role'],

'fname' => $\_POST['fname'],

'sname' => $\_POST['sname'],

'gender' => $\_POST['gender'],

'phone' => $\_POST['phone'],

'dob' => $\_POST['dob'],

'address' => $\_POST['address'],

'city' => $\_POST['city'],

'country' => $\_POST['country'],

'postcode' => $\_POST['postcode']

];

$stmt->execute($criteria);

$stmt = $pdo->prepare('UPDATE office\_staff SET officeid = :officeid

WHERE staffid = ' . $\_POST['submit\_changes']);

$criteria = [

'officeid' => $\_POST['office']

];

$stmt->execute($criteria);

}

function deleteStaff($pdo, $staffid){

$query = $pdo->query('SELECT COUNT(meetingid) FROM meetings where staffid = ' . $staffid);

$fetch = $query->fetch();

if($fetch[0] == '0'){

$delete = $pdo->query('DELETE FROM office\_staff WHERE staffid = ' . $staffid);

$delete = $pdo->query('DELETE FROM staff WHERE staffid = ' . $staffid);

}else{

echo 'That staff member still has scheduled meetings, please delete the meetings before deleting the staff member.<br><br>';

}

}

function newStaff($pdo){

$officeQuery = $pdo->query('SELECT officeid, office\_name FROM offices;');

echo'<form action="index.php" method="POST" enctype="multipart/form-data">

<label for="role"> Choose role:

<select name="role">

<option value="senior instructor">Senior Instructor</option>

<option value="instructor">Instructor</option>

<option value="admin">Administrative Staff</option>

<option value="other">Other</option>

</select><br>

<label for="fname">First Name:

<input name= "fname" type="text"><br>

<label for="sname">Surname:

<input name= "sname" type="text"><br>

<label for="gender">Gender:

<input name= "gender" type="text"><br>

<label for="phone">Phone Number:

<input name= "phone" type="text"><br>

<label for="dob">DOB:

<input name= "dob" type="text">

<br><label for="office"> Choose office:

<select name="office">';

foreach($officeQuery as $row){

echo'<option value="' . $row['officeid'] . '" ';

echo '>' . $row['office\_name'] . '</option>';

}

echo '</select><br>

<label for="address">Address:

<input name= "address" type="text"><br>

<label for="city">City:

<input name= "city" type="text"><br>

<label for="country">Country:

<input name= "country" type="text"><br>

<label for="postcode">Postcode:

<input name= "postcode" type="text"><br>

<button type="submit" name="submit\_new\_staff">Submit</button><br>

</form>';

}

function submitNewStaff($pdo){

$stmt = $pdo->prepare('INSERT INTO staff (role, fname, sname, gender, phone, dob, address, city, country, postcode)

VALUES (:role, :fname, :sname, :gender, :phone, :dob, :address, :city, :country, :postcode)');

$criteria = [

'role' => $\_POST['role'],

'fname' => $\_POST['fname'],

'sname' => $\_POST['sname'],

'gender' => $\_POST['gender'],

'phone' => $\_POST['phone'],

'dob' => $\_POST['dob'],

'address' => $\_POST['address'],

'city' => $\_POST['city'],

'country' => $\_POST['country'],

'postcode' => $\_POST['postcode']

];

$stmt->execute($criteria);

$stmt = $pdo->prepare('SELECT staffid FROM staff WHERE role = :role AND fname = :fname AND sname = :sname AND gender = :gender

AND phone = :phone AND dob = :dob AND address = :address AND city = :city AND country = :country AND postcode = :postcode');

$stmt->execute($criteria);

$staffid = $stmt->fetch();

$office\_staff\_insert = $pdo->prepare('INSERT INTO office\_staff (officeid, staffid)

VALUES (:officeid, :staffid)');

$criteria = [

'officeid' => $\_POST['office'],

'staffid' => $staffid['staffid']

];

$office\_staff\_insert->execute($criteria);

}

?>

### 9.4.4 addTable.php

<?php

class addTable {

public $headings;

public $rows = [];

public function setHeadings($headings) {

$this->headings = $headings;

}

public function addRow($row) {

$this->rows[] = $row;

}

public function getHTML() {

$result = '<table>';

$result = $result . '<thead>';

$result = $result . '<tr>';

foreach ($this->headings as $heading) {

$result = $result . '<th>' . $heading . '</th>';

}

$result = $result . '</tr>';

$result = $result . '</thead>';

$result = $result . '<tbody>';

foreach ($this->rows as $row) {

$result = $result . '<tr>';

foreach ($row as $cell) {

$result = $result . '<td>' . $cell . '</td>';

}

$result = $result . '</tr>';

}

$result = $result . '</tbody>';

$result = $result . '</table>';

return $result;

}

}

?>

### 9.3.5 footer.php

<?php

// The footer lists my name and my email address across every page.

echo' <footer>

<p>Website created by Warren Haskins</p>

<p>Email:whaskins@hotmail.co.uk</p>

</footer>

</body>

</html>';

?>

### 9.3.6 styles.css

body{

background-color: #fff;

}

header{

/\* Here the header is given size, colour and a border. The border radius property lets us curve the border's edges and is used throughout the site. \*/

background-color:#C0C0C0;

border-style:solid;

border-radius:25px;

width:98%;

margin-left:auto;

margin-right:auto;

text-align:center;

clear:both;

padding-top:1px;

}

.mainbody{

/\* This section gives the largest box its features and makes sure it sits to the left of the sidebar. \*/

background-color:white;

margin-left:1%;

padding-left:0%;

font-family:sans-serif;

margin-top:1%;

margin-bottom:1%;

clear:both;

text-align:center;

min-height:350px;

}

table{

margin-left:auto;

margin-right:auto;

}

footer{

/\* The footer is just like the header but at the bottom of the page. \*/

background-color:#C0C0C0;

border-style:solid;

border-radius:25px;

width:98%;

margin-left:auto;

margin-right:auto;

text-align:center;

clear:both;

padding-top:1px;

}

.navigation{

text-align:center;

}