University of Bolton

Student Name	Thomas Plumpton
Student Number	1500936
Student Email	TP2AMT@Bolton.ac.uk

Programme	Computing
Module Code	CPU6007
Module Title	Advanced Database Systems
Date of Submission	7th January 2018
Word Count	6227 (1460 Briefing Document)

Tutor Name	Andrew Parker
Assignment Title	University Accommodation Office Case Study

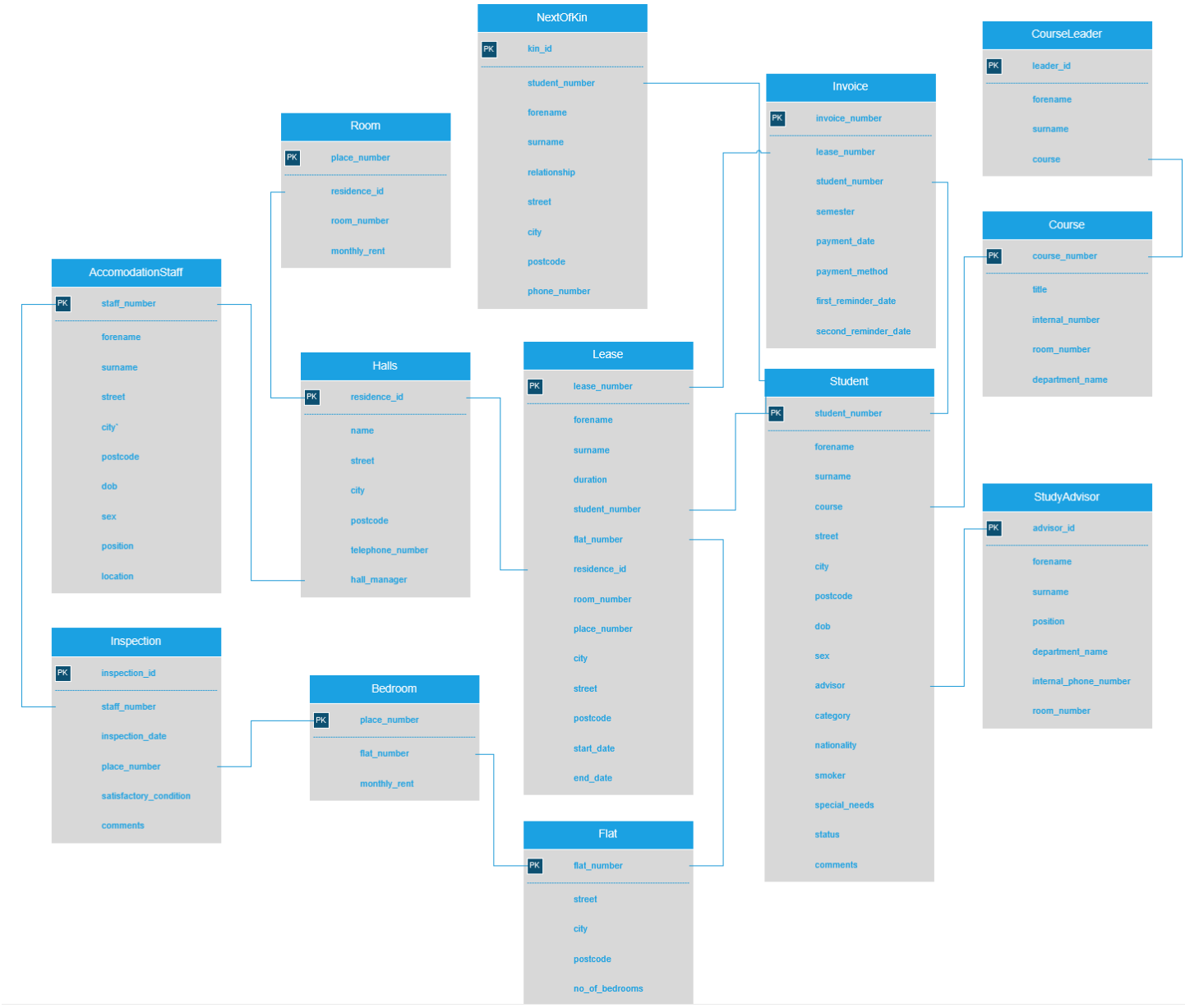
Table of Contents

[1] Database Design

The following section illustrates the design process for the database structure and the data types that were chosen.

[1.1] Table Structure & Data Types

The table structure was decided upon and visualised using Microsoft Visio. The screenshot bellows shows the table structure and relationships.



[1.2] Relational Database Management System

A Relational Database Management System (RDBMS) is a type of DBMS that uses a relational model for its databases. Some of the more popular RDBMS's include Microsoft Access, SQL Server, Oracle Database, MySQL and PostgreSQL. A DBMS is essentially

a ‘type’ of database which defines how the data is stored. It allows users of the system to manipulate the data or manage the structure of the database. (IBM, 2017)

The database created for this assignment uses the MySQL, the most popular Open Source RDBMS. (Oracle, 2017)

[2] Data Definition

This section describes that data definition, illustrating the creation of the database relations and implementing the data types, constraints and field lengths.

[2.1] Creating the Database

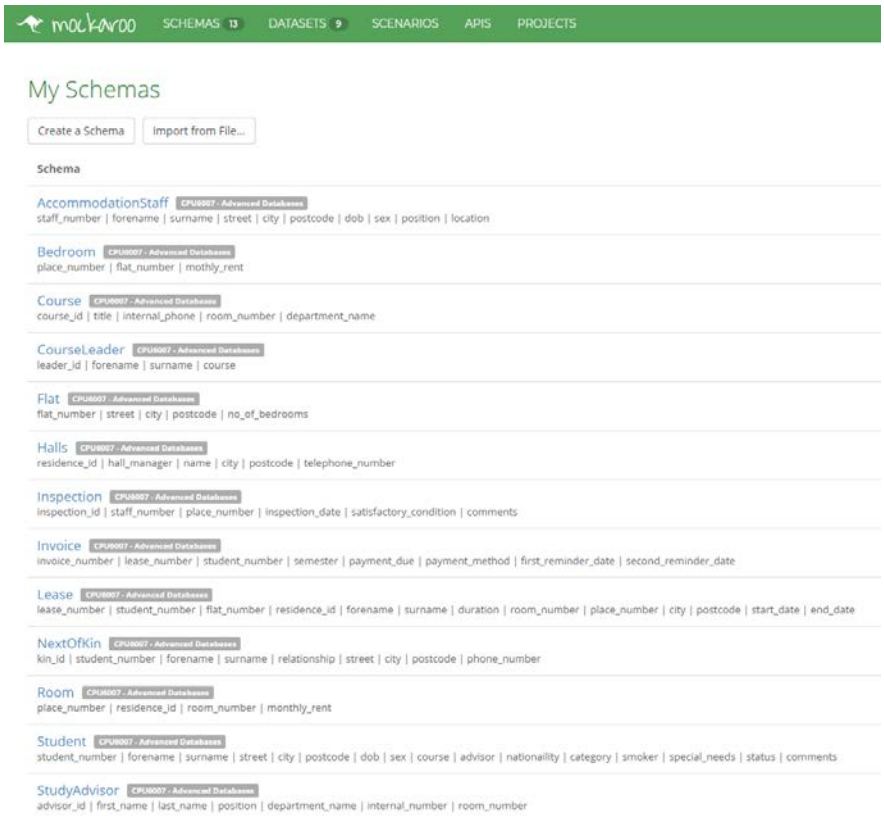
```
CREATE TABLE Course (
  course_id INTEGER NOT NULL AUTO_INCREMENT,
  title VARCHAR(50) NOT NULL,
  internal_phone VARCHAR(11),
  room_number INTEGER NOT NULL,
  department_name VARCHAR(50),
  PRIMARY KEY (course_id)
) AUTO_INCREMENT = 1;

CREATE TABLE CourseLeader (
  leader_id INTEGER NOT NULL AUTO_INCREMENT,
  forename VARCHAR(25) NOT NULL,
  surname VARCHAR(25) NOT NULL,
  course INTEGER NOT NULL,
  PRIMARY KEY (leader_id),
  FOREIGN KEY (course) REFERENCES Course(course_id)
) AUTO_INCREMENT = 1;

CREATE TABLE StudyAdvisor (
  advisor_id INTEGER NOT NULL AUTO_INCREMENT,
  forename VARCHAR(25) NOT NULL,
  surname VARCHAR(25) NOT NULL,
  position VARCHAR(50) NOT NULL,
  department_name VARCHAR(25),
  internal_number VARCHAR(11),
  room_number VARCHAR(4),
  PRIMARY KEY (advisor_id)
) AUTO_INCREMENT = 1;

CREATE TABLE Student (
  student_number INTEGER NOT NULL AUTO_INCREMENT,
  forename VARCHAR(25) NOT NULL,
  surname VARCHAR(25) NOT NULL,
  street VARCHAR(50),
  city VARCHAR(50),
  postcode VARCHAR(9),
  dob DATE,
  sex VARCHAR(15),
  course INTEGER NOT NULL,
  advisor INTEGER NOT NULL,
```

```
  nationality VARCHAR(50),
  category VARCHAR(50),
  smoker BOOLEAN,
  special_needs VARCHAR(150),
  status VARCHAR(50),
  comments VARCHAR(256),
  PRIMARY KEY (student_number),
  FOREIGN KEY (course) REFERENCES Course(course_id),
  FOREIGN KEY (advisor) REFERENCES StudyAdvisor(advisor_id)
) AUTO_INCREMENT = 1;

CREATE TABLE Flat (
  flat_number INTEGER NOT NULL AUTO_INCREMENT,
  street VARCHAR(50),
  city VARCHAR(50),
  postcode VARCHAR(9),
  no_of_bedrooms INTEGER,
  PRIMARY KEY (flat_number)
) AUTO_INCREMENT = 1;

CREATE TABLE AccommodationStaff (
  staff_number INTEGER NOT NULL AUTO_INCREMENT,
  forename VARCHAR(25) NOT NULL,
  surname VARCHAR(25) NOT NULL,
  street VARCHAR(50),
  city VARCHAR(50),
  postcode VARCHAR(9),
  dob DATE,
  sex VARCHAR(15),
  position VARCHAR(25),
  location VARCHAR(25),
  PRIMARY KEY (staff_number)
) AUTO_INCREMENT = 1;

CREATE TABLE Halls (
  residence_id INTEGER NOT NULL AUTO_INCREMENT,
  hall_manager INTEGER NOT NULL,
  name VARCHAR(50) NOT NULL,
  city VARCHAR(50),
  street VARCHAR(50),
  postcode VARCHAR(9),
  telephone_number VARCHAR(11),
  PRIMARY KEY (residence_id),
  FOREIGN KEY (hall_manager) REFERENCES
AccommodationStaff(staff_number)
) AUTO_INCREMENT = 1;

CREATE TABLE Lease (
  lease_number INTEGER NOT NULL AUTO_INCREMENT,
  student_number INTEGER NOT NULL,
  flat_number INTEGER NOT NULL,
```

```
  residence_id INTEGER NOT NULL,
  forename VARCHAR(50) NOT NULL,
  surname VARCHAR(50) NOT NULL,
  duration BIGINT,
  room_number INTEGER,
  place_number INTEGER NOT NULL,
  city VARCHAR(50),
  street VARCHAR(50),
  postcode VARCHAR(9),
  start_date DATETIME,
  end_date DATETIME,
  PRIMARY KEY (lease_number),
  FOREIGN KEY (student_number) REFERENCES
Student(student_number),
  FOREIGN KEY (flat_number) REFERENCES Flat(flat_number),
  FOREIGN KEY (residence_id) REFERENCES Halls(residence_id)
) AUTO_INCREMENT = 1;

CREATE TABLE Invoice (
  invoice_number INTEGER NOT NULL AUTO_INCREMENT,
  lease_number INTEGER NOT NULL,
  student_number INTEGER NOT NULL,
  semester VARCHAR(20),
  payment_due DECIMAL(13, 2),
  payment_method VARCHAR(15),
  first_reminder_date DATE,
  second_reminder_date DATE,
  PRIMARY KEY (invoice_number),
  FOREIGN KEY (lease_number) REFERENCES Lease(lease_number),
  FOREIGN KEY (student_number) REFERENCES Student(student_number)
) AUTO_INCREMENT = 1;

CREATE TABLE Bedroom (
  place_number INTEGER NOT NULL AUTO_INCREMENT,
  flat_number INTEGER NOT NULL,
  monthly_rent DECIMAL(13, 2),
  PRIMARY KEY (place_number),
  FOREIGN KEY (flat_number) REFERENCES Flat(flat_number)
) AUTO_INCREMENT = 1;

CREATE TABLE Inspection (
  inspection_id INTEGER NOT NULL AUTO_INCREMENT,
  staff_number INTEGER NOT NULL,
  place_number INTEGER NOT NULL,
  inspection_date DATE,
  satisfactory_condition BOOLEAN,
  comments VARCHAR(100),
  PRIMARY KEY (inspection_id),
  FOREIGN KEY (staff_number) REFERENCES
AccommodationStaff(staff_number),
  FOREIGN KEY (place_number) REFERENCES Bedroom (place_number)
```

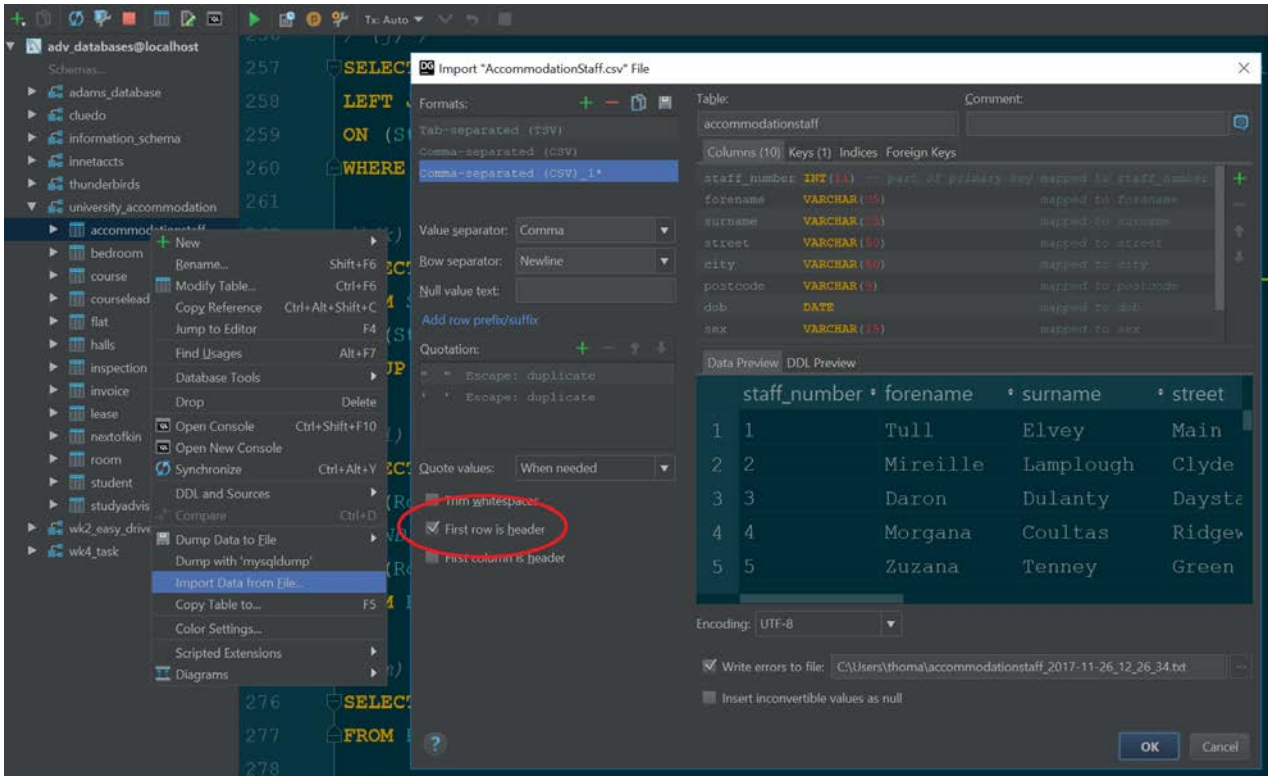
```
) AUTO_INCREMENT = 1;

CREATE TABLE Room (
  place_number INTEGER NOT NULL AUTO_INCREMENT,
  residence_id INTEGER NOT NULL,
  room_number INTEGER NOT NULL,
  monthly_rent DECIMAL(13, 2),
  PRIMARY KEY (place_number),
  FOREIGN KEY (residence_id) REFERENCES Halls(residence_id)
) AUTO_INCREMENT = 1;

CREATE TABLE NextOfKin (
  kin_id INTEGER NOT NULL AUTO_INCREMENT,
  student_number INTEGER NOT NULL,
  forename VARCHAR(25),
  surname VARCHAR(25),
  relationship VARCHAR(25),
  street VARCHAR(50),
  city VARCHAR(50),
  postcode VARCHAR(9),
  phone_number VARCHAR(11),
  PRIMARY KEY (kin_id),
  FOREIGN KEY (student_number) REFERENCES Student(student_number)
) AUTO_INCREMENT = 1;
```

[2.2] Generating the Test Data



[2.3] Inserting the Test Data

After generating the date from Mockaroo, it was to be inserted into the relevant tables from the downloaded .CSV files. The initial method used was using the '**Import Data From File…**' method in JetBrains DataGrip. This was achieved by right clicking a table in a schema, selecting the option and choosing the respective .CSV file. I had to ensure the 'First row is header' option was ticked to that the headings on the first row were not inserted as data.



The second method was using the 'LOAD DATA INFILE' SQL command. This was used simply as an efficiency and quality of life method. It allowed easy recreation of the database as with one key-press, all of the create table commands and insert data commands could be executed at once if structural table changes needed to be made.

```
LOAD DATA INFILE 'C:/University/Databases/Course.csv'
INTO TABLE Course FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE 'C:/University/Databases/CourseLeader.csv'
INTO TABLE CourseLeader FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE 'C:/University/Databases/StudyAdvisor (10).csv'
INTO TABLE StudyAdvisor FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE 'C:/University/Databases/Student.csv'
INTO TABLE Student FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE 'C:/University/Databases/Flat.csv'
INTO TABLE Flat FIELDS TERMINATED BY ',' IGNORE 1 LINES;
```

```
LOAD DATA INFILE 'C:/University/Databases/AccommodationStaff.csv'
INTO TABLE AccommodationStaff FIELDS TERMINATED BY ',' IGNORE 1
LINES;

LOAD DATA INFILE 'C:/University/Databases/Halls (10).csv'
INTO TABLE Halls FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE 'C:/University/Databases/Lease.csv'
INTO TABLE Lease FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE 'C:/University/Databases/Invoice.csv'
INTO TABLE Invoice FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE 'C:/University/Databases/Bedroom.csv'
INTO TABLE Bedroom FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE 'C:/University/Databases/Inspection.csv'
INTO TABLE Inspection FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE 'C:/University/Databases/Room.csv'
INTO TABLE Room FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE 'C:/University/Databases/NextOfKin.csv'
INTO TABLE NextOfKin FIELDS TERMINATED BY ',' IGNORE 1 LINES;
```

[3] Data Manipulation

This section of the report includes descriptions and evidence of all the relevant DML (Data Manipulation Language) statements that were executed upon the database.

[3.1] Testing CRUD Operations

1) The following INSERT statement shows an example of a CREATE operation.

```
INSERT INTO Student VALUES (1000, 'Thomas',
'Plumpton', '20 Murrayfield', 'Bamford', 'OL11 5UQ',
'1997/03/02', 'Male', 12, 5, 'White British', 'Third
Year Undergraduate', 0, 'N/A', 'Placed', 'Wants 95%
in everything');
```

The DataGrip command console output shows the success of the query.

```
sql> INSERT INTO Student VALUES (1001, 'Thomas', 'Plumpton', '20 Murrayfield',
 'Bamford', 'OL11 5UQ', '1997/03/02', 'Male', 12, 5, 'White British', 'Third Year
 Undergraduate', 0, 'N/A', 'Placed', 'Wants 95% in everything')
[2017-12-02 09:47:04] 1 row affected in 15ms
```

2) The following SELECT statement shows an example of a READ operation.

```
SELECT student_number, forename, surname, sex,
category FROM Student WHERE student_number = '1001';
```

The following table shows the output when selecting the ID of the test records inserted above.

	student_number	forename	surname	sex	category
1	1001	Thomas	Plumpton	Male	Third Year Undergraduate

3) The following UPDATE statement shows an example of an UPDATE operation.

```
UPDATE Student SET category = 'Postgraduate' WHERE
student_number = 1001;
```

The DataGrip command console output shows the success of the query.

```
sql> UPDATE Student SET category = 'Postgraduate' WHERE student_number = 1001
[2017-12-02 10:12:38] 1 row affected in 12ms
```

The screenshot below shows the updated SELECT statement on the same record. It shows that Student 1001's category has been changed from 'Third Year Undergraduate' to 'Postgraduate'.

	student_number	forename	surname	sex	category
1	1001	Thomas	Plumpton	Male	Postgraduate

4) The following DELETE statement shows an example of a DELETE operation.

```
DELETE FROM Student WHERE student_number = 1001;
```

The DataGrip command console output shows the success of the query.

```
sql> DELETE FROM Student WHERE student_number = 1001
[2017-12-02 10:19:18] 1 row affected in 13ms
```

To prove that the record had been successfully deleted, the SELECT statement for the READ operation was re-run. The DataGrip console output shows that zero records were returned after running the DELETE statement.

```
sql> SELECT student_number, forename, surname, sex, category FROM Student WHERE
 student_number = '1001'
[2017-12-02 10:19:59] 0 rows retrieved in 56ms (execution: 16ms, fetching: 40ms)
```

[3.2] Querying the Database

A) Present a report listing the Manager's name and telephone number for each hall of residence.

```
SELECT Halls.residence_id AS 'Hall ID',
AccommodationStaff.forename,
AccommodationStaff.surname, telephone_number
FROM Halls INNER JOIN AccommodationStaff
ON (AccommodationStaff.staff_number =
Halls.hall_manager);
```

This query joins the AccommodationStaff and Halls tables on the condition that the manager of the halls matches an ID from the AccommodationStaff table. The result bellows shows the report of all the Hall Managers for each of the 10 Halls of Residence.

	`Hall ID`	forename	surname	telephone_number
1	1	Nan	Dwine	8273072473
2	2	Lance	O'Driscole	9964484656
3	3	Sampson	Champ	2153084633
4	4	Odo	Turton	2811329228
5	5	Daisy	Tolley	5939885286
6	6	Alvis	Greer	3895922628
7	7	Grantley	Forlong	1119706030
8	8	Mic	Angeli	3234578896
9	9	Arni	Selman	9604428509
10	10	Bondie	Cartman	5556575389

B) Present a report listing the names and student numbers of students with the details of their lease agreements.

```
SELECT Student.student_number, Student.forename,
Student.surname, Lease.lease_number,
Lease.room_number, Lease.end_date
FROM Student INNER JOIN Lease
ON (Student.student_number = Lease.student_number);
```

This query joins the Student and Lease tables on the condition that the Student ID on the Lease Agreement matches that of a student. It then prints out any relevant information of the lease. There were 999 results from students, the first 10 are shown below.

	student_num...	forename	surname	lease_number	room_number	end_date
1	1	Sherilyn	Gurnee	128	15	2017-05-18 19:50:44
2	4	Evy	Aprahamian	40	47	2017-09-11 09:21:22
3	4	Evy	Aprahamian	355	97	2017-02-25 10:14:44
4	4	Evy	Aprahamian	441	99	2017-05-07 19:56:07
5	5	Rhona	Musk	133	85	2017-07-20 00:38:00
6	5	Rhona	Musk	164	9	2017-03-20 14:03:26
7	5	Rhona	Musk	436	66	2017-10-24 07:06:59
8	6	Ruprecht	Firebrace	285	92	2017-05-29 02:00:42
9	7	Lorilee	Nail	462	18	2016-12-17 14:52:23
10	7	Lorilee	Nail	475	37	2017-04-03 18:57:27

C) Display the details of lease agreements that include the Summer Semester.

```
SELECT Lease.lease_number, Lease.room_number,
Lease.student_number, Lease.end_date
FROM Lease INNER JOIN Invoice
ON (Lease.lease_number = Invoice.lease_number)
WHERE Invoice.semester = 'Summer Semester';
```

This query joins the Lease and the Invoice table on the condition that the unique Lease ID matches that of the Lease ID on an Invoice. It then checks to see if that invoice was for a lease agreement in the Summer Semester. There were 331 results, the first 10 are shown below.

	lease_number	room_number	student_number	end_date
1	324	99	570	2017-08-10 06:27:39
2	965	71	808	2017-10-17 09:58:16
3	873	86	440	2017-08-30 12:33:18
4	911	18	497	2017-02-28 04:08:59
5	402	75	392	2017-01-21 17:43:16
6	177	46	661	2017-08-10 21:58:48
7	655	87	181	2017-06-27 11:50:30
8	324	99	570	2017-08-10 06:27:39
9	193	5	491	2017-04-04 16:38:58
10	636	12	215	2017-05-24 18:31:40

D) Display the details of the total rent paid by a given student.

```
SELECT Student.student_number, Student.forename,
Student.surname, SUM(Invoice.payment_due) AS 'Total
Rent Paid'
FROM Student INNER JOIN Invoice
ON (Student.student_number = Invoice.student_number)
GROUP BY Student.student_number;
```

This query joins the Student and Invoices tables on the condition that the Student ID on the invoice matches that from the student table. The query selects the cumulative sum of the payments from the invoices and groups the results by Student ID. The first 10 results are shown below.

	student_number	forename	surname	`Total Rent Paid`
1	3	Margaretta	Jencken	1338.05
2	7	Lorilee	Nail	1090.71
3	10	Dyna	Playhill	562.16
4	11	Giuseppe	Kremer	1539.79
5	13	Willette	Duckering	2464.51
6	15	Jordan	Dat	1034.73
7	19	Selena	Knevett	1518.64
8	20	Jaymie	Hawkin	821.57
9	21	Brendis	Leppington	1138.15
10	22	Chas	Law	1515.25

E) Present a report on students that have not paid their invoices by a given date.

```
SELECT Student.student_number, Student.forename,
Student.surname,
Invoice.first_reminder_date AS 'Due Date',
Invoice.payment_due
FROM Student INNER JOIN Invoice
ON (Student.student_number = Invoice.student_number)
WHERE Invoice.first_reminder_date > '2017-08-22';
```

This query joins the Student and Invoice tables where the Student ID's match. It then filters the results to only include those who have received a reminder for late payment by the 22/08/2017. 221 results are returned whose reminder dates are after the 22/08/2017. The first 10 results are shown below.

	student_number	forename	surname	`Due Date`	payment_due
1	673	Sayres	Meechan	2017-09-09	983.40
2	832	Rory	MacNeil	2017-08-28	425.41
3	994	Dominic	Rogier	2017-10-14	598.49
4	984	Tove	Naisey	2017-10-08	277.73
5	574	Tandi	Ryal	2017-10-31	340.29
6	561	Cari	Prowting	2017-08-26	746.39
7	786	Brant	Dallewater	2017-09-24	692.82
8	111	Judon	MacKereth	2017-09-13	712.06
9	355	Federico	Stang-Gjer...	2017-08-28	714.31
10	481	Leigh	Founds	2017-11-03	241.69

F) Display the details of flat inspections where the property was found to be in an unsatisfactory condition.

```
SELECT * FROM Inspection WHERE
satisfactory_condition = '0';
```

This query simply selects all the information from the Inspection tables where the satisfactory_condition column = 0. This column is a boolean where 0 is False and 1 is True. Therefore, the results will show all properties that have failed their inspections. There were 492 results, the first 10 of which are shown below.

	inspection_id	staff_number	place_number	inspection_date	satisfactory...	comments
1	1	406	855	2017-06-20	0	Vestibulum...
2	3	477	522	2017-09-20	0	Aenean sit...
3	4	902	456	2016-12-17	0	Morbi port...
4	8	245	746	2017-03-03	0	Nam tristi...
5	9	950	101	2017-06-10	0	Duis aliqu...
6	11	265	331	2017-07-12	0	Lorem ipsu...
7	14	776	882	2017-08-12	0	Nulla ac e...
8	15	639	792	2017-10-12	0	Sed sagitt...
9	16	659	86	2016-12-21	0	Cum sociis...
10	23	301	846	2017-01-12	0	Donec semp...

G) Present a report of the names and student numbers of students with their room number and place number in a particular hall of residence.

```
SELECT Room.residence_id, Lease.student_number,
Lease.forename, Lease.surname, Room.place_number,
Room.room_number
FROM Room
```

```
INNER JOIN Lease ON (Room.residence_id =
Lease.residence_id) GROUP BY Lease.student_number;
```

This query joins the Lease and the Room tables. It matches them on their respective unique Hall ID's. Finally, it groups the results by the Student ID in the Lease table. All 647 students who are staying in a Halls of Residence are shown with the relevant information. The first 10 are shown below.

	residence_id	student_number	forename	surname	place_number	room_number
1	9	1	Michell	Summers	12	22
2	10	4	Rice	MacAloren	3	31
3	4	5	Vale	Paggitt	10	72
4	4	6	Aurie	Yeowell	10	72
5	10	7	Katherina	Jonuzi	3	31
6	2	9	Shandie	Duffer	24	45
7	6	11	Sari	Cleare	7	73
8	1	12	Marcus	Beyne	2	34
9	6	13	Mercie	Higbin	7	73
10	2	16	Sidnee	Corbridge	24	45

H) Present a report listing the details of all students currently on the waiting list for accommodation that is, not placed.

```
SELECT * FROM Student WHERE status = 'Waiting';
```

This query simply selects all the information from the Student table and filters the result by the status column where it is 'Waiting'. The first 10 results are shown below, none of the students in the report are placed in accommodation.

	student_number	forename	surname	street	city
1	3	Margaretta	Jencken	Old Shore	Perry Crofts
2	4	Evy	Aprahamian	Debs	Fleet
3	9	Lou	Winters	Old Gate	Warrenpoint
4	12	Tamra	Poley	Summer Ridge	Sunderland
5	19	Selena	Knevett	Bowman	Manchester
6	22	Chas	Law	Susan	Old Radnor
7	23	Jozef	Haythornthwaite	Barnett	Stanhope
8	24	Bartie	Rutter	Spenser	Wesham
9	25	Hayes	Nourse	Hallows	Rugeley
10	28	Nedda	Philcock	Longview	Bangor

I) Display the total number of students in each student category.

```
SELECT category, COUNT(*) AS Amount FROM Student
GROUP BY category;
```

This query selects the Student Category and counts the number of tuples returned. But it then groups by the category so it returns the total number of students in each one.

	category	Amount
1	First Year Undergraduate	185
2	Masters Student	164
3	PHD Student	171
4	Postgraduate	158
5	Second Year Undergraduate	168
6	Third Year Undergraduate	153

J) Present a report of the names and student numbers for all students who have not supplied details of their next-of-kin.

```
SELECT Student.student_number, Student.forename,
Student.surname, NextOfKin.kin_id FROM Student
LEFT JOIN NextOfKin
ON (Student.student_number =
NextOfKin.student_number)
WHERE NextOfKin.student_number IS NULL;
```

This query uses a left join on the Student and NextOfKin tables in order to select all the Student tuples that are not associated with a NextOfKin tuple. It then checks to see if the student number belonging the next of kin is null meaning the student has not provided details of their next-of-kin. The first 10 results are shown below, as you can see, all of the kin_id's are null.

	student_number	forename	surname	kin_id
1	3	Margaretta	Jencken	<null>
2	4	Evy	Aprahamian	<null>
3	10	Dyna	Playhill	<null>
4	11	Giuseppe	Kremer	<null>
5	15	Jordan	Dat	<null>
6	16	Haslett	Franchi	<null>
7	21	Brendis	Leppington	<null>
8	22	Chas	Law	<null>
9	23	Jozef	Haythornthwaite	<null>
10	27	Colan	Fishwick	<null>

K) Display the name and internal telephone number of the Advisor of Studies for a particular student.

```
SELECT StudyAdvisor.forename, StudyAdvisor.surname,
StudyAdvisor.internal_number, Student.student_number
FROM StudyAdvisor INNER JOIN Student
ON (StudyAdvisor.advisor_id = Student.advisor)
GROUP BY Student.student_number;
```

This query selects the relevant Advisor of Studies information and join the StudyAdvisor and Student tables on the condition that the advisor ID marches. It then groups by Student ID resulting in 999 results (one for each student). The first 10 are shown below.

	forename	surname	internal_number	student_number
1	Robb	Belleny	7317586692	1
2	Robb	Belleny	7317586692	3
3	Mari	Yaknov	2838995732	4
4	Florina	Cinderey	1587182373	5
5	Troy	Stoade	5037330496	6
6	Florina	Cinderey	1587182373	7
7	Moses	Lapides	1355260421	8
8	Mari	Yaknov	2838995732	9
9	Florina	Cinderey	1587182373	10
10	Eugen	Titmuss	8116227541	11

L) Display the minimum, maximum, and average monthly rent for rooms in halls of residence.

```
SELECT
MIN(Room.monthly_rent) AS 'Minimum',
ROUND(AVG(Room.monthly_rent), 2) AS 'Average',
MAX(Room.monthly_rent) AS 'Maximum'
FROM Room;
```

This query simply selects the minimum, average and maximum monthly rents from the Room table. The average function is wrapped in a round to 2 decimal places for greater precision.

	Minimum	Average	Maximum
1	100.30	297.25	499.49

M) Display the total number of places in each hall of residence.

```
SELECT Room.residence_id AS 'Hall ID', COUNT(*) AS
'Number of Rooms'
FROM Room GROUP BY Room.residence_id;
```

This query counts all the tuples from the rooms tables but groups them by the Hall ID. The results show the number of places in each Halls of Residence.

	`Hall ID`	`Number of Rooms`
1	1	92
2	2	103
3	3	91
4	4	110
5	5	122
6	6	97
7	7	86
8	8	95
9	9	108
10	10	96

N) Display the staff number, name, age, and current location of all members of the accommodation staff who are over 60 years old on the day the report is run.

```
SELECT staff_number AS 'Staff ID',
FLOOR(ABS(DATEDIFF(AccommodationStaff.dob,
CURRENT_DATE)) / 365.25636) AS Age, location
FROM AccommodationStaff HAVING Age > 60;
```

The final query is a bit more complicated. It first selects the Staff ID from the AccommodationStaff table. Then the next selection is a calculation of age from the stored date of birth value. Starting from the inside out, it uses the DATEDIFF function and passes it the staff members Date of Birth and the System Time. This calculates the time difference between the two. It then divides this value by the number of days in a year, which is roughly 365.25363 (to 5 decimal places). This is then wrapped in the ABS function which returns the absolute values to eliminate a negative value. Finally, the value if floored to round it down to the nearest integer. This calculation is stored in the Age alias which is reference in the final HAVING clause to filter the results by those that are greater than 60 years of age. The first 10 results are shown below.

	`Staff ID`	Age	location
1	19	66	Halls of Residence
2	20	65	Accommodation Office
3	23	64	Accommodation Office
4	39	67	Halls of Residence
5	67	65	Accommodation Office
6	73	65	Accommodation Office
7	74	61	Accommodation Office
8	75	65	Halls of Residence
9	79	64	Accommodation Office
10	91	66	Halls of Residence

[4] Database Briefing

The Director of the University Accommodation Office has requested a briefing document that outlines the various methods that clients can use to access the database and the technology required to host the database. The following section includes the requested information and takes into consideration the existence of multiple platforms when accessing the database.

[4.1] Accessing the Database

There are various methods in which clients can access the University Accommodation database. This section outlines three suitable and common approaches that should be taken into consideration.

[4.1.1] Web Integration

One of the more popular methods for accessing databases is via a front-end webpage via an Internet browser. A web environment would be established involving the client connecting to a web server via HTTP over a TCP/IP Network. The web server would serve static HTML files to the client which would be rendered in the browser. Server-side scripting languages such as PHP or JavaScript can then be used to establish a connection to the database and send data to the page. Client-side scripts a long with HTML & CSS styling will display the data in a readable format for the user. The use of a CGI is one method by which the web server can obtain data from a database. One of the main advantages for this is it can be written in any programming language.

[4.1.2] Purpose Built Application

Another approach is the development of a purpose-built application that will suite the University Accommodations use-case. This could be developed in an Object Orientated Programming (OOP) Language such as Java. Libraries such as JDBC can be used in

order to communicate with the database. A Java EE Server would be used with an EJB Container which is an architecture for program components that run in the server parts of a client/server model. This would mean that downloaded data would be stored in POJOs.

[4.1.3] Command Line Interface (CLI)

Another option is to access the database via a CLI. The command line is easier to develop for and is great for automation. The problem with command line interfaces is that they are harder to learn, particularly for those that do not have a technical background. This is because unlike a GUI, a CLI does not use graphical elements such as windows, icons, menus and pointer in order to enhance the user experience and make it easier for the user to navigate about the program.

[4.2] Hosting Technologies

The database has been developed using the DBMS 'MySQL'. A server is required to host the database so that it can be accessed by the application. The database server needs to be able to communicate with the web-server in order for server-side scripts to connect to the database and read data from it. It also need to have regular communication with any backup servers to ensure that is not lost should an accident occur.

The second is a web-server that will be used to deliver or 'serve' web pages. The hardware specification of this server is subjective to the chosen access method. If the web-based application method is chosen, then the web-server will need to have a mid-tier spec. An adequate amount of RAM should be installed on the server as it used to hold the relevant temporary data when it is running multiple processes at once, such as scripts. A high-end CPU is not required as the web-server will not be receiving a high amount of simultaneous traffic. Therefore, a relatively low-end, budget friendly CPU will suffice.

The third and final server isn't necessarily a requirement, but is strongly recommended. This server would ideally be independent from the others and would be used to store back-ups, snapshots and replica sets of the database. The hardware specifications of the server need only be basic as processing speed isn't paramount and can be done slowly overnight. The major high-spec component would be the network bandwidth and transfer speed.

The aforementioned servers can be purchased and hosted at the University itself, or virtual server space can be purchased on a subscription plan with a provider such as Amazon. Amazon offer AWS plans of varying hardware specifications and support. One of the major advantages of choosing an online cloud solution such as AWS is that all of the setup and configuration is handled by the provider. This ultimately saves money on both initial setup costs and those involved in maintaining the servers as they are handled by the provider.

[4.3] Security & Maintainability

[4.3.1] Logical Security

Logical security is non-tangible and consists of software safeguards put in place to protect an organisations systems. User accounts are an essential part of any security

system. They provide the clients with unique usernames and a secure password that allows entry to the computer system. Passwords are encrypted into a hash when stored in a database and while being transferred over the network. User object are serialised upon arrival on the client-side, decrypted, authenticated and the de-serialised. This increases security as potential hackers cannot see the passwords in plain text. It also prevents man-in-the-middle attacks, which is when an attacker intercepts a message on the network before it reaches its destination.

A hierarchy of access levels is another important implementation which limits the data in which the users can see. This means that users can only access data that they require, subject to their role. This reduces the risk of white collar crime whereby an employee steals company data to misuse it for their own purpose.

[4.3.2] Physical Security

It is also important to consider physical security. For example, the physical devices and computer hardware can be secured to their environment to prevent theft. This includes Kensington locks, doors secured with Yale locks, RFID cards or biometric systems. Biometrics include devices such as fingerprint and retinal scanners to gain entry to secure rooms.

Another security measure is employing security personnel to work on-site and to verify the identity of everybody the enters and leaves the premises. This adds an extra layer of physical security which makes it very difficult for an attacker to physically steal data from the premises.

A very secure but expensive implementation is a Demilitarised Zone (DMZ) which would separate the internal LAN from any external un-trusted networks. It is common practice to make sure all Web, Mail, FTP and VoIP servers are kept in the DMZ, meaning any external communication is routed through the firewall for increased security. A DMZ includes both physical and logical security.

[4.3.3] Network Security

The implementation of an SSL Certificate on the web-site will ensure that all data will be served over a secure port via HTTPS, a secure version of the HTTP protocol. This means that all sensitive information, such as that belonging to a user account, will be encrypted to increase security.

Another suggestion that suits the Universities use-case is the implementation of an Intranet or Extranet. An Intranet is a LAN of computers that do not have access to the Internet. This vastly increases security as external hackers have no means to remotely access the data on the computers connected to the internal network.

However, considering that the University may need Internet access, an extranet could be implemented instead. Similar to an intranet, an extranet is built upon it and is partially accessible to authorised outsiders. This could include business suppliers and partners. The network will be protected by a firewall to help control access to the between the intranet and the Internet.

[4.4] Access Strategy Recommendation

We recommend hosting a MySQL database on an in-house server. The database would be accessed from a front-end web page. Modern web-frameworks are easy to set-up and expand and can be accessed from any platform or operating system with a web browser and Internet access. Although the initial setup costs will be higher than opting for an online web service such as AWS, it gives the administrators full control over the hardware and network security meaning it can be configured to work as fast and as secure as possible for the University.

Staff will be provided with user accounts which they can use to access the database management page on campus. It is therefore recommended to use an campus-wide extranet to keep the data as secure as possible.

Finally, a backup server should be configured to backup the databases every day which will be overseen by the administrative personnel. It is recommended that this server is kept offsite. It this is the case, the database server will communicate with the backup server through the firewall. If it is not feasible to keep it offsite, it is recommended to keep it as far as possible from the main server room to reduce the probability of data loss in a natural disaster or theft.

[5] Glossary of Terms

Term	Definition
AWS	Amazon Web Services. A comprehensive, evolving cloud computing platform provided by Amazon.com. Web services are sometimes called cloud services or remote computing services.
CGI	Common Gateway Interface. A standard protocol for web servers to execute programs that execute like Console applications (also called Command-line interface programs) running on a server that generates web pages dynamically.
CLI	Command Line Interface. A user interface to a computer's operating system or an application in which the user responds to a visual prompt by typing in a command on a specified line, receives a response back from the system, and then enters another command, and so forth.
CPU	Central Processing Unit. The CPU is also known as the processor or microprocessor. The CPU is responsible for executing a sequence of stored instructions called a program.
CSS	Cascading Style Sheets. A style sheet language used for describing the presentation of a document written in a markup language.
DBMS	Database Management System. A computer-software application that interacts with end-users, other applications, and the database itself to capture and analyse data. A general-purpose DBMS allows the definition, creation, querying, update, and administration of databases.
DMZ	De-militarised Zone. A physical or logical subnetwork that contains and exposes an organization's external-facing services to an untrusted network, usually a larger network such as the Internet.
DOS	The term DOS can refer to any operating system, but it is most often used as a shorthand for MS-DOS (Microsoft disk operating system). Originally developed by Microsoft for IBM, MS-DOS was the standard operating system for IBM-compatible personal computers.

EJB	Enterprise Java Beans. An architecture for setting up program components, written in the Java programming language, that run in the server parts of a computer network that uses the client/server model.
Extranet	An intranet that can be partially accessed by authorized outside users, enabling businesses to exchange information over the Internet in a secure way.
Firewall	A part of a computer system or network which is designed to block unauthorized access while permitting outward communication.
FTP	File Transfer Protocol. Used to transfer files between computers on a network. You can use FTP to exchange files between computer accounts, transfer files between an account and a desktop computer, or access online software archives.
GUI	Graphical User Interface. A type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation.
HTML	Hyper-Text Markup Language. A standardized system for tagging text files to achieve font, colour, graphic, and hyperlink effects on World Wide Web pages.
HTTP(S)	Hyper-Text Transfer Protocol (Secure). The underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. HTTPS is the secure version of HTTP that uses SSL to encrypt data.
Intranet	A local or restricted communications network, especially a private network created using World Wide Web software.
Java	A general-purpose computer programming language designed to produce programs that will run on any computer system.
JavaScript	An object-oriented computer programming language commonly used to create interactive effects within web browsers.
JDBC	Java Database Connection. An application programming interface (API) for the programming language Java, which defines how a client may access a database
LAN	Local Area Network. A network that connects computers and other devices in a relatively small area, typically a single building or a group of buildings.
Linux	An open-source operating system modelled on UNIX.
MySQL	An open source relational database management system (RDBMS) based on Structured Query Language (SQL)
OOP	Object Orientated Programming. A programming language model organized around objects rather than "actions" and data rather than logic.
OSX	OS X is version 10 of the Apple Macintosh operating system. OS X was described by Apple as its first "complete revision" of the OS since the previous version is OS 9, with a focus on modularity so that future changes would be easier to incorporate.
POJO	Plain Old Java Object. A normal Java object class (that is, not a JavaBean, EntityBean etc.) and does not serve any other special role nor does it implement any special interfaces of any of the Java frameworks.
PHP	Hyper-Text Processor. A script language and interpreter that is freely available and used primarily on Linux Web servers.

RAM	Random Access Memory. A type of computer memory that can be accessed randomly; that is, any byte of memory can be accessed without touching the preceding bytes.
RFID	Radio-frequency Identification. A technology that incorporates the use of electromagnetic or electrostatic coupling in the radio frequency (RF) portion of the electromagnetic spectrum to uniquely identify an object, animal, or person.
Server	A computer or computer program which manages access to a centralized resource or service in a network.
SSL	Secure Socket Layer. The standard security technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browsers remain private and integral.
TCP/IP	Transmission Control Protocol / Internet Protocol. A set of rules that governs the connection of computer systems to the Internet.
UNIX	An operating system analogous to DOS and Windows, supporting multiple concurrent users.
VoIP	Voice over Internet Protocol. The set of rules that makes it possible to use the Internet for telephone or videophone communication.
Windows	A GUI operating system for personal computers.

[6] Bibliography

hcidata.ingo, 2016. *?NET - What Do They Mean?.* [Online]
Available at: http://www.hcidata.info/inet.htm
[Accessed 02 12 2017].

Ian, 2016. *What is an RDBMS?.* [Online]
Available at: http://database.guide/what-is-an-rdbms/
[Accessed 09 12 2017].

IBM, 2017. *What is a database management system?.* [Online]
Available at:
https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zmiddbmg/zmiddle_46.htm
[Accessed 09 12 2017].

Matrejek, K., n.d. *CGI Programming 101.* [Online]
Available at: http://www.cgi101.com/book/intro.html
[Accessed 30 11 2017].

Medic, M., 2014. *Creating and using CRUD stored procedures.* [Online]
Available at: https://www.sqlshack.com/creating-using-crud-stored-procedures/
[Accessed 02 12 2017].

Oracle, 2017. *1.3.1 What is MySQL?.* [Online]
Available at: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html
[Accessed 09 12 2017].

Rose, M., n.d. *DMZ (demilitarized zone).* [Online]
Available at: http://searchsecurity.techtarget.com/definition/DMZ
[Accessed 01 12 2017].

UoB, 2017. *Advanced Database Assignment.* [Online]
Available at: http://moodle2.bolton.ac.uk/mod/resource/view.php?id=355523
[Accessed 25 10 2017].

UoB, 2017. *Assignment Case Study.* [Online]
Available at: http://moodle2.bolton.ac.uk/mod/resource/view.php?id=362612
[Accessed 25 10 2017].

[7] Transaction Log

```
CREATE TABLE Course (
  course_id INTEGER NOT NULL AUTO_INCREMENT,
  title VARCHAR(50) NOT NULL,
  internal_phone VARCHAR(11),
  room_number INTEGER NOT NULL,
  department_name VARCHAR(50),
  PRIMARY KEY (course_id)
) AUTO_INCREMENT = 1;

CREATE TABLE CourseLeader (
  leader_id INTEGER NOT NULL AUTO_INCREMENT,
  forename VARCHAR(25) NOT NULL,
  surname VARCHAR(25) NOT NULL,
  course INTEGER NOT NULL,
  PRIMARY KEY (leader_id),
  FOREIGN KEY (course) REFERENCES Course(course_id)
) AUTO_INCREMENT = 1;

CREATE TABLE StudyAdvisor (
  advisor_id INTEGER NOT NULL AUTO_INCREMENT,
  forename VARCHAR(25) NOT NULL,
```

```
  surname VARCHAR(25) NOT NULL,
  position VARCHAR(50) NOT NULL,
  department_name VARCHAR(25),
  internal_number VARCHAR(11),
  room_number VARCHAR(4),
  PRIMARY KEY (advisor_id)
) AUTO_INCREMENT = 1;

CREATE TABLE Student (
  student_number INTEGER NOT NULL AUTO_INCREMENT,
  forename VARCHAR(25) NOT NULL,
  surname VARCHAR(25) NOT NULL,
  street VARCHAR(50),
  city VARCHAR(50),
  postcode VARCHAR(9),
  dob DATE,
  sex VARCHAR(15),
  course INTEGER NOT NULL,
  advisor INTEGER NOT NULL,
  nationality VARCHAR(50),
  category VARCHAR(50),
  smoker BOOLEAN,
  special_needs VARCHAR(150),
  status VARCHAR(50),
  comments VARCHAR(256),
  PRIMARY KEY (student_number),
  FOREIGN KEY (course) REFERENCES Course(course_id),
  FOREIGN KEY (advisor) REFERENCES StudyAdvisor(advisor_id)
) AUTO_INCREMENT = 1;

CREATE TABLE Flat (
```

```
  flat_number INTEGER NOT NULL AUTO_INCREMENT,
  street VARCHAR(50),
  city VARCHAR(50),
  postcode VARCHAR(9),
  no_of_bedrooms INTEGER,
  PRIMARY KEY (flat_number)
) AUTO_INCREMENT = 1;

CREATE TABLE AccommodationStaff (
  staff_number INTEGER NOT NULL AUTO_INCREMENT,
  forename VARCHAR(25) NOT NULL,
  surname VARCHAR(25) NOT NULL,
  street VARCHAR(50),
  city VARCHAR(50),
  postcode VARCHAR(9),
  dob DATE,
  sex VARCHAR(15),
  position VARCHAR(25),
  location VARCHAR(25),
  PRIMARY KEY (staff_number)
) AUTO_INCREMENT = 1;

CREATE TABLE Halls (
  residence_id INTEGER NOT NULL AUTO_INCREMENT,
  hall_manager INTEGER NOT NULL,
  name VARCHAR(50) NOT NULL,
  city VARCHAR(50),
  street VARCHAR(50),
  postcode VARCHAR(9),
  telephone_number VARCHAR(11),
  PRIMARY KEY (residence_id),
```

```
  FOREIGN KEY (hall_manager) REFERENCES AccommodationStaff(staff_number)
) AUTO_INCREMENT = 1;

CREATE TABLE Lease (
  lease_number INTEGER NOT NULL AUTO_INCREMENT,
  student_number INTEGER NOT NULL,
  flat_number INTEGER NOT NULL,
  residence_id INTEGER NOT NULL,
  forename VARCHAR(50) NOT NULL,
  surname VARCHAR(50) NOT NULL,
  duration BIGINT,
  room_number INTEGER,
  place_number INTEGER NOT NULL,
  city VARCHAR(50),
  street VARCHAR(50),
  postcode VARCHAR(9),
  start_date DATETIME,
  end_date DATETIME,
  PRIMARY KEY (lease_number),
  FOREIGN KEY (student_number) REFERENCES Student(student_number),
  FOREIGN KEY (flat_number) REFERENCES Flat(flat_number),
  FOREIGN KEY (residence_id) REFERENCES Halls(residence_id)
) AUTO_INCREMENT = 1;

CREATE TABLE Invoice (
  invoice_number INTEGER NOT NULL AUTO_INCREMENT,
  lease_number INTEGER NOT NULL,
  student_number INTEGER NOT NULL,
  semester VARCHAR(20),
  payment_due DECIMAL(13, 2),
  payment_method VARCHAR(15),
```

```
  first_reminder_date DATE,
  second_reminder_date DATE,
  PRIMARY KEY (invoice_number),
  FOREIGN KEY (lease_number) REFERENCES Lease(lease_number),
  FOREIGN KEY (student_number) REFERENCES Student(student_number)
) AUTO_INCREMENT = 1;

CREATE TABLE Bedroom (
  place_number INTEGER NOT NULL AUTO_INCREMENT,
  flat_number INTEGER NOT NULL,
  monthly_rent DECIMAL(13, 2),
  PRIMARY KEY (place_number),
  FOREIGN KEY (flat_number) REFERENCES Flat(flat_number)
) AUTO_INCREMENT = 1;

CREATE TABLE Inspection (
  inspection_id INTEGER NOT NULL AUTO_INCREMENT,
  staff_number INTEGER NOT NULL,
  place_number INTEGER NOT NULL,
  inspection_date DATE,
  satisfactory_condition BOOLEAN,
  comments VARCHAR(100),
  PRIMARY KEY (inspection_id),
  FOREIGN KEY (staff_number) REFERENCES AccommodationStaff(staff_number),
  FOREIGN KEY (place_number) REFERENCES Bedroom (place_number)
) AUTO_INCREMENT = 1;

CREATE TABLE Room (
  place_number INTEGER NOT NULL AUTO_INCREMENT,
  residence_id INTEGER NOT NULL,
  room_number INTEGER NOT NULL,
```

```
  monthly_rent DECIMAL(13, 2),
  PRIMARY KEY (place_number),
  FOREIGN KEY (residence_id) REFERENCES Halls(residence_id)
) AUTO_INCREMENT = 1;

CREATE TABLE NextOfKin (
  kin_id INTEGER NOT NULL AUTO_INCREMENT,
  student_number INTEGER NOT NULL,
  forename VARCHAR(25),
  surname VARCHAR(25),
  relationship VARCHAR(25),
  street VARCHAR(50),
  city VARCHAR(50),
  postcode VARCHAR(9),
  phone_number VARCHAR(11),
  PRIMARY KEY (kin_id),
  FOREIGN KEY (student_number) REFERENCES Student(student_number)
) AUTO_INCREMENT = 1;

/* Load Data From CSV */
LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/Course.csv'
INTO TABLE Course FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/CourseLeader.csv'
INTO TABLE CourseLeader FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/StudyAdvisor (10).csv'
INTO TABLE StudyAdvisor FIELDS TERMINATED BY ',' IGNORE 1 LINES;
```

LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/Student.csv'

INTO TABLE Student FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/Flat.csv'

INTO TABLE Flat FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/AccommodationStaff.csv'

INTO TABLE AccommodationStaff FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/Halls (10).csv'

INTO TABLE Halls FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/Lease.csv'

INTO TABLE Lease FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/Invoice.csv'

INTO TABLE Invoice FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/Bedroom.csv'

INTO TABLE Bedroom FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/Inspection.csv'

INTO TABLE Inspection FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/Room.csv'

```
INTO TABLE Room FIELDS TERMINATED BY ',' IGNORE 1 LINES;

LOAD DATA INFILE '/Users/thoma/Dropbox (University)/Year 3/Semester 1/CPU6007 - Advanced Database Systems (Andrew)/Assignment/datasets/NextOfKin.csv'
INTO TABLE NextOfKin FIELDS TERMINATED BY ',' IGNORE 1 LINES;
/* Queries */

/*(a)*/
SELECT Halls.residence_id AS 'Hall ID', AccommodationStaff.forename,
AccommodationStaff.surname, telephone_number
FROM Halls INNER JOIN AccommodationStaff
ON (AccommodationStaff.staff_number = Halls.hall_manager);

/*(b)*/
SELECT Student.student_number, Student.forename, Student.surname,
Lease.lease_number, Lease.room_number, Lease.end_date
FROM Student INNER JOIN Lease
ON (Student.student_number = Lease.student_number);

/*(c)*/
SELECT Lease.lease_number, Lease.room_number, Lease.student_number,
Lease.end_date
FROM Lease INNER JOIN Invoice
ON (Lease.lease_number = Invoice.lease_number)
WHERE Invoice.semester = 'Summer Semester';

/*(d)*/
SELECT Student.student_number, Student.forename, Student.surname,
SUM(Invoice.payment_due) AS 'Total Rent Paid'
FROM Student INNER JOIN Invoice
ON (Student.student_number = Invoice.student_number)
GROUP BY Student.student_number;
```

```
/*(e)*/
SELECT Student.student_number, Student.forename, Student.surname,
Invoice.first_reminder_date AS 'Due Date', Invoice.payment_due
FROM Student INNER JOIN Invoice
ON (Student.student_number = Invoice.student_number)
WHERE Invoice.first_reminder_date > '2017-08-22';

/*(f)*/
SELECT * FROM Inspection WHERE satisfactory_condition = '0';

/*(g)*/
SELECT Halls.residence_id, Lease.student_number, Lease.forename, Lease.surname,
Room.place_number, Room.room_number
FROM Halls
INNER JOIN Lease ON (Halls.residence_id = Lease.residence_id)
INNER JOIN Room ON (Room.residence_id = Lease.residence_id)
GROUP BY Lease.student_number;

SELECT Room.residence_id, Lease.student_number, Lease.forename, Lease.surname,
Room.place_number, Room.room_number
FROM Room
INNER JOIN Lease ON (Room.residence_id = Lease.residence_id) GROUP BY
Lease.student_number;

/*(h)*/
SELECT * FROM Student WHERE status = 'Waiting';

/*(i)*/
SELECT category, COUNT(*) AS Amount FROM Student GROUP BY category;

/*(j)*/
```

```
SELECT Student.student_number, Student.forename, Student.surname,
NextOfKin.kin_id FROM Student

LEFT JOIN NextOfKin

ON (Student.student_number = NextOfKin.student_number)

WHERE NextOfKin.student_number IS NULL;


/*(k)*/

SELECT StudyAdvisor.forename, StudyAdvisor.surname, StudyAdvisor.internal_number,
Student.student_number

FROM StudyAdvisor INNER JOIN Student

ON (StudyAdvisor.advisor_id = Student.advisor)

GROUP BY Student.student_number;


/*(l)*/

SELECT

MIN(Room.monthly_rent) AS 'Minimum',

ROUND(AVG(Room.monthly_rent), 2) AS 'Average',

MAX(Room.monthly_rent) AS 'Maximum'

FROM Room;


/*(m)*/

SELECT Room.residence_id AS 'Hall ID', COUNT(*) AS 'Number of Rooms'

FROM Room GROUP BY Room.residence_id;


/*(n)*/

SELECT staff_number AS 'Staff ID',

FLOOR(ABS(DATEDIFF(AccommodationStaff.dob, CURRENT_DATE)) / 365.25636) AS
Age, location

FROM AccommodationStaff HAVING Age > 60;
```