F28DM Assignment 1

Database design & Implementation

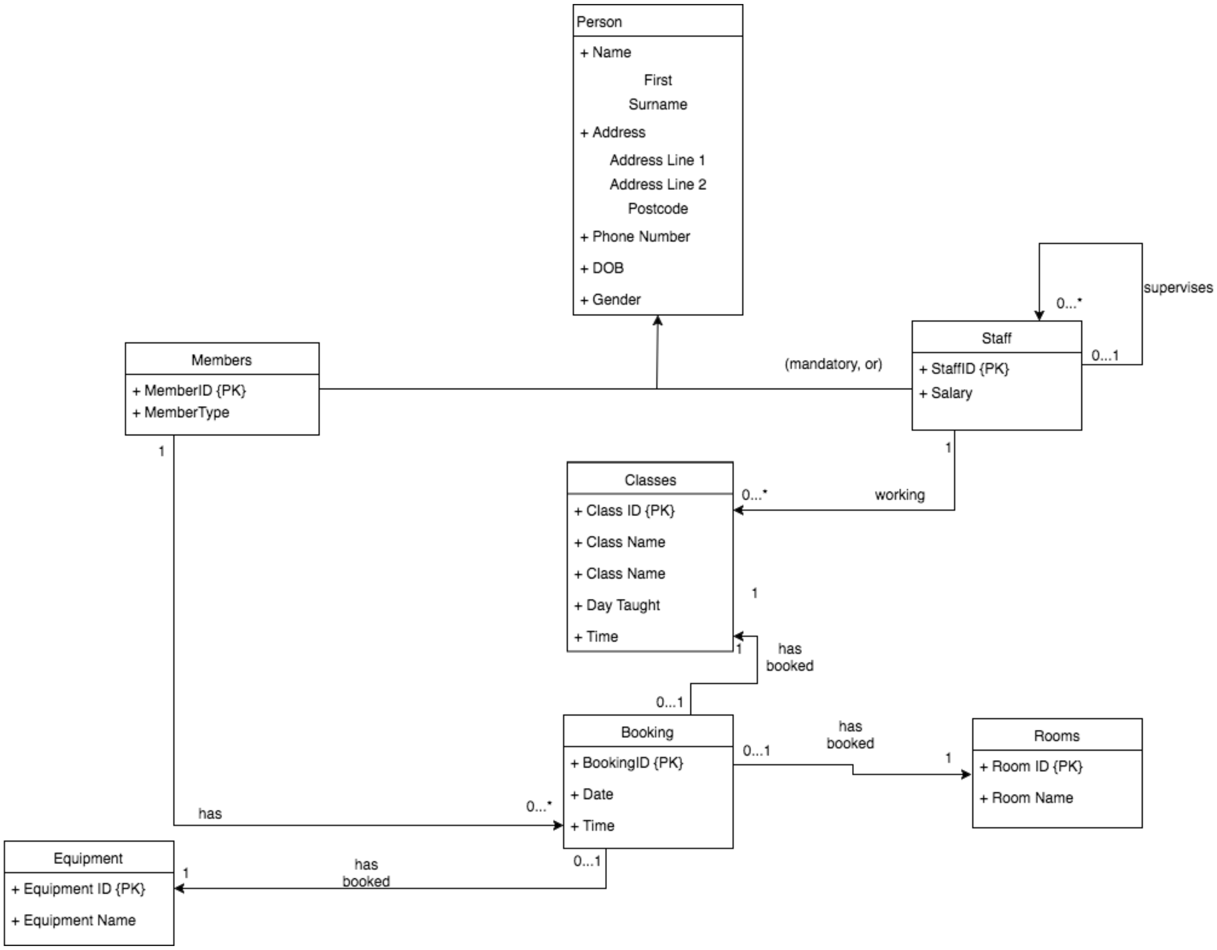
Group 14

Submitted by: Mamta Sofat(H00236213), Nyal Sadiq(H00245545), Mark Gordon(H00228757), Matthew Reilly(H00219672)

T1 Extended Scenario and Conceptual Model

A sports centre needs a booking system to manage members and allow them to book classes, rooms and equipment. The centre is only accessible if a person is either a member of a sports group or a member of staff. A staff member may supervises other staff members. A staff member may or may not work for classes. A member can make one or more bookings. A member can book a class, room, and equipment separately and is assigned with a separate booking ID each time he/she wants to book a class or room or equipment.

ER Diagram



T2 Translation into Relational Schema

In the ER diagram, a person within the sports centre must either be a member, or a member of staff. So the relational schema sorts people into one of two tables, the “Members” table or the “Staff” table. In the ER diagram a member can book one of three things: a class, a room, or a piece of equipment. When a member books something they are assigned a booking ID which is held in the “Booking” entity. This was translated into the relational schema by creating three seperate relations - “RoomBookings”, “ClassBookings”, and “EquipmentBookings”. Each of these tables related the booking ID to the piece of equipment/room/class that they were for. The booking ID must be linked back to the member that made the booking, so a “MemberBookings” table was created which related the relevant member ID to each booking ID. In the ER diagram a member of staff can supervise zero to many other members of staff. This was captured by creating the “StaffSupervisors” table which relates staff ID’s to their supervisors staff ID. Members of staff are able to teach multiple classes. This is represented in the relational schema by the “StaffInstructors” table which relates the staff members staff ID to the class ID of the class that they are teaching.

The “MemberType” part of the member entity was placed in a seperate relation, linking memberID’s to membership types.

Data Dictionary

Members

P/F	Field Name	Label	Type(Width)	Value Codes
Primary Key	member_id	Member ID	integer(11)	auto_increment
	firstName	First Name	string(<20)	none
	surname	Surname	string(<20)	none
	addressLine1	Address Line 1	string(<40)	none
	addressLine2	Address Line 2	string(<40)	none
	postcode	Postcode	string(9)	none
	type	Type	integer(1.0)	1=Gold 2=Silver 3=Bronze
	contact_number	Contact_Number	integer(11.0)	none
	dateofBirth	Date of Birth	date(11)(dd/mm/yyyy)	none
	gender	Gender	integer(1.0)	1=Male 2=Female

Member Bookings

P/F	Field Name	Label	Type(Width)	Value Codes
Foreign Key	bookingID	Booking ID	integer(11)	auto_increment
Foreign Key	memberID	Member ID	integer(11)	auto_increment

Membership Types

P/F	Field Name	Label	Type(Width)	Value Codes
Foreign Key	memberID	Member ID	integer(11)	auto_increment
	type	Type of Membership	set('Gold', 'Silver', 'Bronze')	none

Staff

P/F	Field Name	Label	Type(Width)	Value Codes
Primary Key	staffID	Staff ID	integer(11)	auto_increment
	firstName	First Name	string(<20)	none
	surname	Surname	string(<20)	none
	addressLine1	Address Line 1	string(<40)	none
	addressLine2	Address Line 2	string(<40)	none
	postcode	Postcode	string(9)	none
	salary	Salary	integer(11)	none
	contact_number	Contact_Number	integer(11.0)	none
	dateofBirth	Date of Birth	date(11)(dd/mm/yyyy)	none
	age	Age	integer(3.0)	none
	gender	Gender	integer(1.0)	1=Male 2=Female

Staff Instructors

P/F	Field Name	Label	Type(Width)	Value Codes
	staffID	Staff ID	integer(11)	auto_increment
	classID	Class ID	integer(11)	auto_increment

Staff Supervisors

P/F	Field Name	Label	Type(Width)	Value Codes

Foreign Key	staffID	Staff ID	integer(11)	auto_increment
	supervisorsStaffID	Supervisors Staff ID	integer(11)	auto_increment

Classes

P/F	Field Name	Label	Type(Width)	Value Codes
Primary Key	classID	Class ID	integer(11)	auto_increment
	className	Name of Class	string(<40)	none
	time	Time of Class	time(11)(hh/mm/ss)	none
	dayTaught	Day of Class	set('Mon', 'Tue', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun’)	none

Class Bookings

P/F	Field Name	Label	Type(Width)	Value Codes
Foreign Key	bookingID	Booking ID	integer(11)	auto_increment
Foreign Key	classID	Class ID	integer(11)	auto_increment

Rooms

P/F	Field Name	Label	Type(Width)	Value Codes
Primary Key	roomID	Room ID	integer(11)	auto_increment
	name_number	Name/Number of Room	string(<20)	none

Room Bookings

P/F	Field Name	Label	Type(Width)	Value Codes
Foreign Key	bookingID	Booking ID	integer(11)	auto_increment
Foreign Key	roomID	Room ID	integer(11)	auto_increment

Equipment

P/F	Field Name	Label	Type(Width)	Value Codes
Primary Key	equipmentID	Equipment ID	integer(11)	auto_increment
	equipmentName	Name of Equipment	string(<40)	none

Equipment Bookings

P/F	Field Name	Label	Type(Width)	Value Codes
Foreign Key	bookingID	Booking ID	integer(11)	auto_increment
Foreign Key	equipmentID	Equipment ID	integer(11)	auto_increment

Bookings

P/F	Field Name	Label	Type(Width)	Value Codes
Primary Key	bookingID	Booking ID	integer(11)	auto_increment
	dateofBooking	Date	date(11)(dd/mm/yyyy)	none
	timeofBooking	Time	time(11)(hh/mm/ss)	none

T3 Implementation of schema into MySql

Members

This script creates a table called “Members” with eight columns. The table is used to store details of each member. The “memberID” column stores a unique auto incremented integer. This column cannot contain null values, every member must have a member ID as it is used as a foreign key in another part of the database. The member's first name, surname and address line details are all stored in varchar columns with a maximum string size of 25 characters. The postcode is stored in a varchar column with a maximum string size of 7 characters. The date of birth column stores a date value and must not be null, storing each member's age is mandatory. Values in the gender column can be one of three items from a set - “M”, “F”, or “Other”. The primary key in this table is the “memberID”, it is used as a foreign key in the “MemberBookings” and “MembershipTypes” tables.

```
CREATE TABLE `Members` (
  `memberID` int(11) NOT NULL AUTO_INCREMENT,
  `firstName` varchar(25) CHARACTER SET utf8 DEFAULT NULL,
  `surname` varchar(25) CHARACTER SET utf8 DEFAULT NULL,
  `addressLine1` varchar(25) CHARACTER SET utf8 DEFAULT NULL,
  `addressLine2` varchar(25) CHARACTER SET utf8 DEFAULT NULL,
  `postcode` varchar(7) CHARACTER SET utf8 DEFAULT NULL,
  `dateofBirth` date NOT NULL COMMENT 'YYYY-MM-DD',
  `gender` set('M','F','Other','') CHARACTER SET utf8 DEFAULT NULL,
  PRIMARY KEY (`memberID`)
  UNIQUE KEY ‘memberID’ (‘memberID’)
) ENGINE=InnoDB AUTO_INCREMENT=47 DEFAULT CHARSET=utf8
```

Staff

The “Staff” table is much the same as the “Members” table although it contains an extra column - “salary”. This is an integer field that contains each member of staff’s salary.

```
CREATE TABLE `Staff` (
  `staffID` int(11) NOT NULL AUTO_INCREMENT,
  `firstName` varchar(25) CHARACTER SET utf8 DEFAULT NULL,
  `surname` varchar(25) CHARACTER SET utf8 DEFAULT NULL,
  `addressLine1` varchar(25) CHARACTER SET utf8 DEFAULT NULL,
  `addressLine2` varchar(25) CHARACTER SET utf8 DEFAULT NULL,
  `postcode` varchar(7) CHARACTER SET utf8 DEFAULT NULL,
  `dateofBirth` date NOT NULL COMMENT 'YYYY-MM-DD',
  `gender` set('M','F','Other','') CHARACTER SET utf8 DEFAULT NULL,
  `salary` int(11) DEFAULT NULL,
  PRIMARY KEY (`staffID`),
  UNIQUE KEY `staffID` (`staffID`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8
```

Bookings

This script creates a table called “Bookings”. It is used to store details of bookings that members have made. The table contains three columns - “bookingID”, “dateofBooking”, and “timeofBooking”. The “BookingID” column is an auto incremented integer column, which cannot be null and is unique. The “dateofBooking” column stores the day, month, and year that a member made the booking. It is stored using the date data type. The “timeofBooking” column stores the time, in 24 hour format, that the booking was made. The primary key in this table is the “bookingID”, it is used in the “ClassBookings”, “RoomBookings”, “EquipmentBookings”, and “MemberBookings” tables.

```
CREATE TABLE `Bookings` (
  `bookingID` int(10) NOT NULL AUTO_INCREMENT COMMENT 'Unique',
  `dateofBooking` date DEFAULT NULL COMMENT 'YYYY-MM-DD',
  `timeofBooking` varchar(5) DEFAULT NULL COMMENT 'HH:MM',
  PRIMARY KEY (`bookingID`),
  UNIQUE KEY `bookingID` (`bookingID`)
) ENGINE=InnoDB AUTO_INCREMENT=21 DEFAULT CHARSET=utf8
```

Classes

This script creates a table called “Classes”. It contains four columns, “ClassID”, “className”, “dayTaught”, and “time”. As with the primary keys in all the other tables, “ClassID” is a unique auto incremented integer. The “className” column stores the title of the class, which will default to NULL if left empty - this allows staff to timetable new class slots before knowing the exact details of them. The “dayTaught” column accepts values from a set of the days of the week. The “time” column is a varchar field where the text should be in the “HH:MM” format.

```
CREATE TABLE `Classes` (
  `classID` int(11) NOT NULL AUTO_INCREMENT,
  `className` varchar(25) DEFAULT NULL,
  `dayTaught` set('Mon','Tue','Wed','Thurs','Fri','Sat','Sun') DEFAULT NULL,
  `time` varchar(5) DEFAULT NULL,
  PRIMARY KEY (`classID`),
  UNIQUE KEY `classID` (`classID`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8
```

ClassBookings

This script creates a table called “ClassBookings”. This table relates booking ID’s to class ID’s. This allows us to store and query all of the bookings for classes. The foreign key constraint for update and delete actions on the parent key, for both foreign keys, is to cascade. This is required because if a member removes their booking for a class, we want to remove it from the list of class bookings. This is the same principle for update actions. Also, if a class is no longer running, then we must remove all corresponding rows from this table.

```
CREATE TABLE `ClassBookings` (
  `bookingID` int(11) NOT NULL,
  `classID` int(11) NOT NULL,
  KEY `Class Bookings` (`bookingID`),
  KEY `Classes` (`classID`),
  CONSTRAINT `Class Bookings` FOREIGN KEY (`bookingID`) REFERENCES `Bookings`
(`bookingID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `Classes` FOREIGN KEY (`classID`) REFERENCES `Classes` (`classID`) ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Equipment

This script creates a table called “Equipment”. This stores details of all the equipment available for members to borrow. Is contains two columns, “EquipmentID” and “equipmentName”.

```
CREATE TABLE `Equipment` (
  `equipmentID` int(11) NOT NULL AUTO_INCREMENT,
  `equipmentName` varchar(25) CHARACTER SET utf8 DEFAULT NULL,
  PRIMARY KEY (`equipmentID`),
  UNIQUE KEY `equipmentID` (`equipmentID`)
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=utf8
```

EquipmentBookings

This script creates a table called “EquipmentBookings”. This table relates booking ID’s to equipmentID’s. This allows us to store and query all of the bookings for equipment. The foreign key constraints were chosen for the same reason as in the “ClassBookings” table.

```
CREATE TABLE `EquipmentBookings` (
  `bookingID` int(10) NOT NULL,
  `equipmentID` int(11) NOT NULL,
  KEY `Equipment Bookings` (`bookingID`),
  KEY `EquipmentID` (`equipmentID`),
  CONSTRAINT `Equipment Bookings` FOREIGN KEY (`bookingID`) REFERENCES
`Bookings` (`bookingID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `EquipmentID` FOREIGN KEY (`equipmentID`) REFERENCES `Equipment`
(`equipmentID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

MembershipTypes

This script creates a table called “MembershipTypes”. This table relates member ID’s to membership types. The foreign key constraints for update and delete actions on the parent key are both cascade. If a member leaves the sports centre, we want to delete all of their data from the system. Similarly if a member ID changes for any reason, we want to reflect this on the foreign key.

```
CREATE TABLE `MembershipTypes` (
  `memberID` int(11) NOT NULL,
  `type` set('Gold','Silver','Bronze','') CHARACTER SET utf8 DEFAULT 'Bronze',
  KEY `MemberID` (`memberID`),
  CONSTRAINT `MemberID` FOREIGN KEY (`memberID`) REFERENCES `Members`
(`memberID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

MemberBookings

This script creates a table called “MemberBookings”. This table relates booking ID’s to member ID’s, so we know what member each booking ID refers to. The foreign key constraints on update or delete actions are to cascade. The reasoning here is the same as the other booking relations, we wish to reflect any changes to bookings on the foreign keys. If a member wishes to cancel their booking, then we must remove all corresponding rows in this table.

```
CREATE TABLE `MemberBookings` (
  `bookingID` int(11) NOT NULL,
  `memberID` int(11) NOT NULL,
  KEY `Bookings` (`bookingID`),
  KEY `Members` (`memberID`),
  CONSTRAINT `Bookings` FOREIGN KEY (`bookingID`) REFERENCES `Bookings`
(`bookingID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `Members` FOREIGN KEY (`memberID`) REFERENCES `Members`
(`memberID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Rooms

This script creates a table called “Rooms”. This table stores information on the rooms that are available for booking by members. It has two columns, “RoomID” and “roomName”. The primary key here. “RoomID”, is an unique auto incremented integer. “roomName” is a varchar column.

```
CREATE TABLE `Rooms` (
  `roomID` int(11) NOT NULL AUTO_INCREMENT,
  `roomName` varchar(25) DEFAULT NULL,
  PRIMARY KEY (`roomID`),
  UNIQUE KEY `roomID` (`roomID`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8
```

RoomBookings

This script creates a table called “RoomBookings”. This table relates booking ID’s to room ID’s. This allows us to store and query all of the bookings made for rooms. The foreign key constraints on update and delete actions to the parent keys are to cascade. This is because we want to reflect any change to a members booking on the foreign keys. If a room is no longer available for booking, then we must remove all corresponding rows in this table. If a member wishes to cancel their booking then we will also remove all corresponding rows in this table.

```
CREATE TABLE `RoomBookings` (
  `bookingID` int(10) NOT NULL,
  `roomID` int(11) NOT NULL,
  KEY `Room Bookings` (`bookingID`),
  KEY `Rooms` (`roomID`),
  CONSTRAINT `Room Bookings` FOREIGN KEY (`bookingID`) REFERENCES `Bookings`
(`bookingID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `Rooms` FOREIGN KEY (`roomID`) REFERENCES `Rooms` (`roomID`) ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

StaffInstructors

This script creates a table called “StaffInstructors”. This table relates staff ID’s to class ID’s. This allows us to store and query the list of what class each member of staff is teaching. The foreign key constraints on update and delete actions to the parent keys are to cascade. This is because if the class is no longer running, then we must delete the row containing that class from the table. Assuming that if an instructor is no longer available then the class is cancelled, then the same applies to the “staffID” foreign key.

```
CREATE TABLE `StaffInstructors` (
  `staffID` int(11) NOT NULL,
  `classID` int(11) NOT NULL,
  KEY `TeachingClassID` (`classID`),
  KEY `InstructorID` (`staffID`),
  CONSTRAINT `InstructorID` FOREIGN KEY (`staffID`) REFERENCES `Staff` (`staffID`) ON
DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `TeachingClassID` FOREIGN KEY (`classID`) REFERENCES `Classes`
(`classID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

StaffSupervisors

This script creates a table called “StaffSupervisors”. This table relates staff ID’s to their supervisors staff ID. This allows us to store a list of staff members and their supervisors which makes accountability simpler when the business encounters problems. The foreign key constraints for update and delete actions on the parent keys are to cascade. We want to reflect any changes to the parent keys on the foreign keys. If a staff member decides to quit, we must remove any rows that correspond to them in this table, as they will not have a supervisor anymore. Similarly, if a supervisor decides to quit, we should remove any rows corresponding to them - the members of staff who used to be supervised them are now without a supervisor. Also, if a member of staff is appointed a new supervisor, we must reflect that update in this table.

```
CREATE TABLE `StaffSupervisors` (
  `staffID` int(11) NOT NULL,
  `supervisorsStaffID` int(11) NOT NULL,
  KEY `StaffID` (`staffID`),
  KEY `SupervisorID` (`supervisorsStaffID`),
  CONSTRAINT `StaffID` FOREIGN KEY (`staffID`) REFERENCES `Staff` (`staffID`) ON
DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `SupervisorID` FOREIGN KEY (`supervisorsStaffID`) REFERENCES `Staff` (`staffID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

T4 Loading Data

Insert Statements

_The data for all the entities: members, staff, classes, booking, rooms and equipment, has been loaded using insert statements. (**Detailed in attached script archive**) A few examples:-

Members:

```
INSERT INTO `Members`(`memberID`, `firstName`, `surname`, `addressLine1`,
`addressLine2`, `postcode`, `dateofBirth`, `gender`) VALUES (1,“Harry”,”Potter”,”121
Godricks Hallow”,”Mars”,”PO6 6XY”,”1991-02-28”,”M”)
```

Staff:

```
INSERT INTO `Staff`(`staffID`, `firstName`, `surname`, `addressLine1`, `addressLine2`,
`postcode`, `dateofBirth`, `gender`, `salary`)  VALUES
('Peter','Stevens','92/6 High Riggs','Edinburgh','EH9 3JH',’1993-08-11’,'M','30,000')
```

Classes:

```
INSERT INTO `Classes`(`classID`, `className`, `dayTaught`, `time`) VALUES
('Karate','Sat',’12:30’)
```

Equipment:

```
INSERT INTO `Equipment`(`equipmentID`, `equipmentName`) VALUES ('Kung Fu suit’,
‘shoes’, ‘Dummies’, ‘Bags’, ‘Holdalls')
```

Rooms:

```
INSERT INTO `Rooms`(`roomID`, `roomName`) VALUES ('Martial Art Hall')
```

Bookings:

```
INSERT INTO `Bookings`(`bookingID`, `dateofBooking`, `timeofBooking`) VALUES ('2017-
02-18',’11:00’)
```

Bulk Loader data

Some data for the entities ‘members’ and ‘staff’ was loaded from bulk loader. We created two separate text files, **members.txt, staff.txt, and equipment.txt (see attached)** and collated the data for each entity. Using the following SQL queries through the linux command line we loaded the data from these files into our database:

For loading members from members.txt:

```
LOAD DATA LOCAL INFILE '/home/cs2/ns53/MySQL/members.txt' REPLACE INTO TABLE Members Fields terminated by ',';
```

For loading staff members from staff.txt:

```
LOAD DATA LOCAL INFILE '/home/cs2/ns53/MySQL/staff.txt' REPLACE INTO TABLE Staff Fields terminated by ',';
```

For loading equipments from equipment.txt:

```
LOAD DATA LOCAL INFILE '/home/cs2/ns53/MySQL/equipment.txt' REPLACE INTO TABLE Equipment Fields terminated by ',';
```

T5 Roles, Permissions and views

Roles

There are four typical users of the Sports Centre database system:

- Manager
- Reception

- Supervisors
- General Staff

The manager of the sports centre is in the most responsible and senior role. They must ensure that the business runs smoothly at all times, therefore they have full access and permissions to the database. They must be able to access every table so that they can monitor the profits the business is making. They will also be able to monitor the progress of all aspects of the business.

The receptionists are the staff who work at the front desk and deal with any customer issues or queries.Their job is to take members bookings, register new members, and also help any instructors with any queries about the classes they teach. So they have full access to the database apart from the “Staff” table and the “Number_of_Supervisors” view, these tables contain information about staff salaries and positions and so must only be accessed by the manager.

The supervisor's job is to monitor all other staff members and ensure they are completing their tasks properly. They have access to all tables apart from the “Members” and “Staff” tables and the “Gold_Members” view. They need this wide access to be able to help their supervisees with any problems. However they do not need to be able to view personal member or staff information as it is beyond the scope of their job. Supervisors are also able to see what members staff are teaching classes, and who other supervisors should be supervising - so that they can ensure staff members are actually completing their tasks.

The General Staff have limited access to the system. They are only allowed to see data from the “Classes”, “Equipment”, and “Rooms” tables. They can also see what class they are teaching, as well as who their supervisor is. They do not need access to any other table due to the nature of their job.

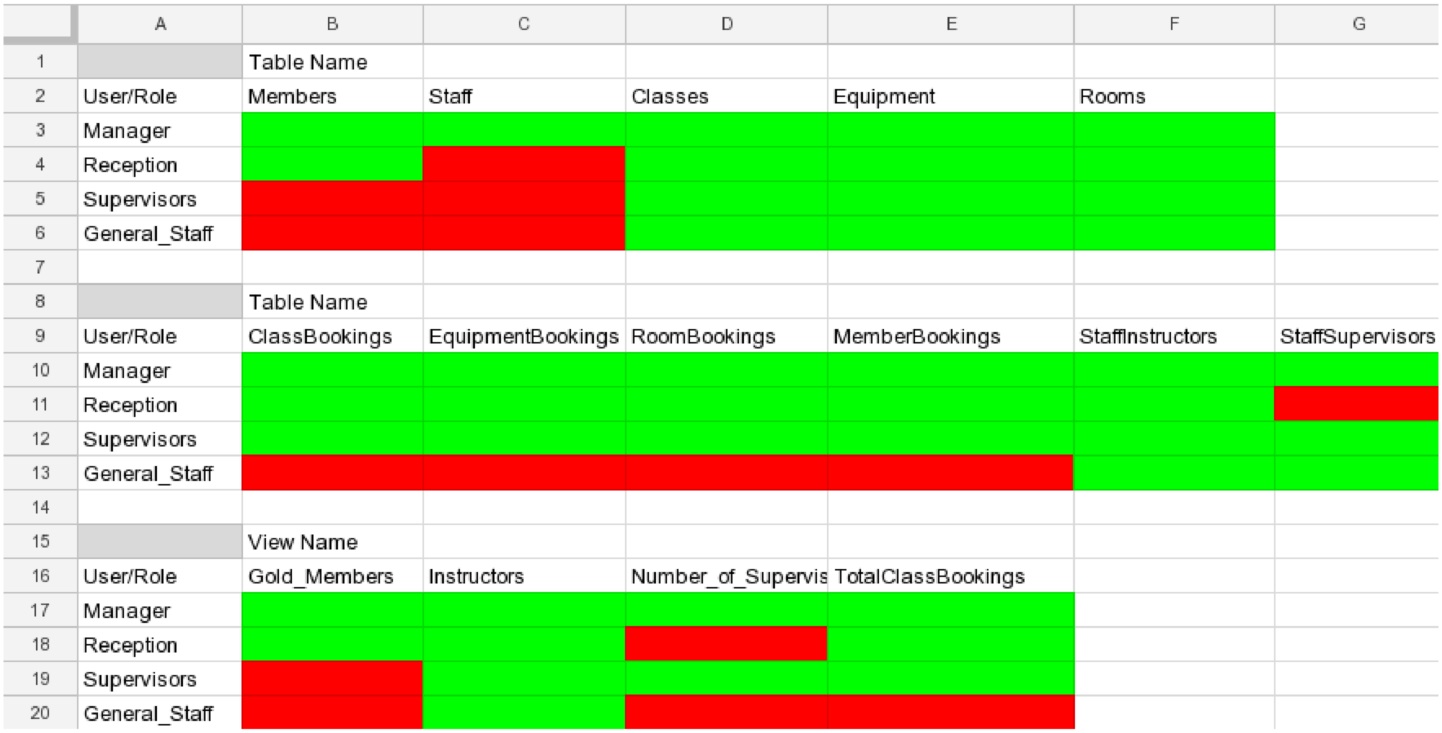
Views

There are 4 views that members of staff - with the appropriate access rights - can select data from.
The “TotalClassBookings” view allows staff to see how many members have booked each class. It displays the number of people attending the class, as well as the class name. This will allow the Sports Centre to see if a class is full and/or allow them to prepare the appropriate lesson materials according to the class size.

The “Gold_Members” view allows staff to see what members hold a gold membership. It displays all of the details of each gold member. This will allow them to check if a member hold the required membership before making perhaps a more exclusive booking.

The “Number_of_Supervisees” view allows supervisors to see how many employees they currently supervise. This is mainly to add a competitive spirit to the centre, the more supervisees the more bragging rights.

The “Instructors” view allows staff to view the names of the classes that they are teaching.



	A	B	C	D	E	F	G
1		Table Name					
2	User/Role	Members	Staff	Classes	Equipment	Rooms	
3	Manager	🟩	🟩	🟩	🟩	🟩	
4	Reception	🟩	🟥	🟩	🟩	🟩	
5	Supervisors	🟥	🟥	🟩	🟩	🟩	
6	General_Staff	🟥	🟥	🟩	🟩	🟩	
7							
8		Table Name					
9	User/Role	ClassBookings	EquipmentBookings	RoomBookings	MemberBookings	StaffInstructors	StaffSupervisors
10	Manager	🟩	🟩	🟩	🟩	🟩	🟩
11	Reception	🟩	🟩	🟩	🟩	🟩	🟥
12	Supervisors	🟩	🟩	🟩	🟩	🟩	🟩
13	General_Staff	🟥	🟥	🟥	🟥	🟩	🟩
14							
15		View Name					
16	User/Role	Gold_Members	Instructors	Number_of_Supervis	TotalClassBookings		
17	Manager	🟩	🟩	🟩	🟩		
18	Reception	🟩	🟩	🟥	🟩		
19	Supervisors	🟥	🟩	🟩	🟩		
20	General_Staff	🟥	🟩	🟥	🟥		

Access Grid

T6 Queries over Database

1. Count number of bookings for class with ID 1:

```
SELECT COUNT(classID) AS NumberOfBookingsForClassID1 FROM
ClassBookings WHERE classID=1;      Using mySQL builtin function
```

Output:

NumberOfBookingsForClassID1

4

2. Return the full name of the Member with the bookingID of 2:

```
SELECT MemberBookings.bookingID, Members.firstName,
Members.surname
FROM Members
INNER JOIN MemberBookings
ON MemberBookings.memberID=Members.memberID
WHERE MemberBookings.bookingID = 2;
```

Output:

bookingID	**firstName**	**surname**
2	Jeff	Stevens

3. Total staff annual salary:

```
SELECT SUM(salary) AS TotalStaffSalary FROM Staff;
```

Output:

TotalStaffSalary
245500

4. Return all the members details, including their membership type, with the MemberID of 4:

```
SELECT Members.*, MembershipTypes.type
FROM Members
INNER JOIN MembershipTypes
ON Members.memberID=MembershipTypes.memberID
WHERE Members.memberID = 4;
```

Output:

member ID	**first Name**	**surname**	**address Line1**	**address Line2**	**postcode**	**dateofBirth YYYY:MM:DD**	**gender**	**type**
4	Harry	Potter	121 Godricks Hallow	Mars	PO6 6XY	1997-12-25	M	Gold

5. Display those members with more than 1 booking:

```
SELECT Members.surname, COUNT(MemberBookings.bookingID) AS
NumberOfBookings
FROM Members
INNER JOIN MemberBookings
ON MemberBookings.memberID=Members.memberID
GROUP BY surname
HAVING COUNT(MemberBookings.memberID) >= 2
```

Output:

surname	**NumberOfBookings**

Khan	2
Palpatine	2
Patinson	2
Potter	2
Radcliffe	2
Weasley	2

6. Average Staff salary:

```
SELECT AVG(salary) AS salaryAverage, MIN(salary) as minSalary,
MAX(salary) AS maxSalary FROM Staff;
```

Output:

salaryAverage	**minSalary**	**maxSalary**
24550.0000	20000	30000

7. Update a members address details:

Before update:

member	**firstNa**	**surname**	**addressLine**	**addressLi**	**postco**	**dateofBirt**	**gend**

ID	me		1	ne2	de	h	er
3	Emperor	Palpatine	Naboo,Death Star II	Coruscant Senate Building	XXXDXXX	1950-02-28	M

```
UPDATE `Members` SET addressLine1="Heriot Watt",
addressLine2="Mary Fergusson Halls", postcode="EH144AS" WHERE
memberID = 3;
```

Output:

memberID	firstName	surname	addressLine1	addressLine2	postcode	dateofBirth	gender
3	Emperor	Palpatine	Heriot Watt	Mary Fergusson Halls	EH144AS	1950-02-28	M

8. Display number of bookings made each day:

```
SELECT COUNT(dateOfBooking) AS numberOfBookings, dateofBooking
FROM Bookings
GROUP BY dateofBooking;
```

Output:

numberOfBookings	dateofBooking
1	2000-05-12
2	2017-02-08

2	2017-02-09
3	2017-02-10
2	2017-02-11
1	2017-02-12
6	2017-02-13
1	2017-02-14

9. Display classes that aren’t Karate or Lightsaber Combat:

```
SELECT * FROM Classes WHERE className NOT IN (‘Karate’,
‘Lightsaber Combat’);
```

Output:

classID	**className**	**dayTaught**	**time**
3	Archery	Tue	16:00
4	Archery	Mon	16:00
5	Fencing	Thurs	10:00
6	Fencing	Wed	15:00
7	Yoga	Tue	18:00
8	Wall Climbing	Mon	11:30

9	Wall Climbing	Fri	16:30
10	Kung Fu	Fri	14:00

10. Delete the member with an ID of 2 bookings from the bookings table:

```
DELETE FROM Bookings
WHERE bookingID IN (SELECT bookingID FROM MemberBookings WHERE
memberID = 2);
```

Output:

This query removes all the bookings made by the member with member ID 2.

11. Show information of staff who are supervisors:

```
SELECT * FROM Staff
WHERE staffID IN (SELECT supervisorsstaffID FROM
StaffSupervisors)
```

Output:

staffID	**firstName**	**surname**	**addressLine1**	**addressLine2**	**postcode**	**dateofBirth**	**gender**	**salary**
1	Peter	Stevens	92/6 High Riggs	Edinburgh	EH9 3JH	1960-10-26	M	30000
5	Alex	Fisher	61 Sing Street	Dublin	WI7 8OK	1950-02-14	M	25000
8	Nick	McCain	2/8 Herriot Street	London	T7 89JK	1992-05-18	M	20000

12. Show information of regular staff (no supervisors):

```
SELECT * FROM Staff
WHERE staffID NOT IN (SELECT supervisorsstaffID FROM
StaffSupervisors)
```

Output:

staffID	firstName	surname	addressLine1	addressLine2	postcode	dateofBirth	gender	salary
2	White	Kinley	Olof Street	Olof	OL2 45N	1900-11-11	M	25000
3	Jammie	Phinnick	76 American Street	Jamaica	PA0 1SJ	2012-12-12	F	20000
4	Georgina	Bush	1 Jordan Lane	DC	WO1 1IK	2011-11-11	F	25000
6	Fanny	Stalin	928/8 Crab Street	Finland	UD6 6IN	1950-09-10	F	20000
7	Martin	Black	289/6 Baron Lane	Mars	U9 99UM	1980-02-10	M	25500
9	Vereena	McCullough	8 Queens Street	Chiswick	TY8 8JN	2001-01-01	F	30000
10	Julius	Patterson	73 Finnish Street	Birmingham	EH3 7YJ	1995-12-10	M	25000

13. Display Instructors view with Staffs names:

```
SELECT S.firstName, S.surname, I.*
FROM Instructors I
INNER JOIN Staff S
WHERE I.staffID = S.staffID
```

Output:

firstName	**surname**	**staffID**	**classID**	**className**
White	Kinley	2	3	Archery
Jammie	Phinnick	3	5	Fencing
Peter	Stevens	1	1	Karate
Peter	Stevens	1	10	Kung Fu
Alex	Fisher	5	8	Wall Climbing
Georgina	Bush	4	7	Yoga

14. Change membership status by name:

```
UPDATE MembershipTypes
SET type = "Gold"
WHERE memberID IN (SELECT memberID
                   FROM Members
                   WHERE surname = "Cena"
                   AND firstName = "John");
```

Output:

This query updates the MembershipTypes relation where the existing attribute type= “Silver” is changed to type = "Gold" for the member with surname = "Cena" and firstName = "John" in the Members relation.

T7 Indexes

An index is created here since the salary column will be queried frequently by managers looking to update a staff members salary.

```
CREATE INDEX sal_index ON Staff(salary);
```

This index is needed since the firstname and surname of members are accessed frequently by instructors in classes to see details of people who have booked their class.

```
CREATE INDEX mName_index ON Members(firstName, surname);
```

An index is created on the dayTaught and time columns as this information is frequently accessed by members of staff to see when classes are on.

```
CREATE INDEX classTD_index ON Classes(dayTaught, time);
```

This index is used since queries from managers looking at the names of each member of staff will be frequent.

```
CREATE INDEX sName_index ON Staff(firstName, surname);
```

T8 Java Application

The JDBCMySQLApp source code is contained in the archive file, the application has also been demonstrated at the lab.

Summary

This course work has been a good learning experience and was completed with each group member contributing an equal amount of work.