1 Introduction

The Structured Query Language (SQL) is the language of databases. SQL was, is, and will stay for the foreseeable future the database language for relational database servers such as IBM DB2, Microsoft SQL Server, MySQL, Oracle, Progress, Sybase Adaptive Server, and dozens of others.

SQL supports a small but very powerful set of statements for manipulating, managing, and protecting data stored in a database. This power has resulted in its tremendous popularity. Almost every database server supports SQL or a dialect of the language. Currently, SQL products are available for every kind of computer, from a small handheld computer to a large server, and for every operating system, including Microsoft Windows, Mac and many UNIX variations.

1.1. What is a Database?

A database is a structured collection of data that is used by the application systems of some given enterprise, and that is managed by a database management system.

For the purpose of this course, think of a database as a collection of tables which are connected to each other. IT Learning Programme (ITLP) in the University of Oxford offers a course on how to design a database. This course is a pre-requisite to this course. However, if you did not attend the database designing course, please read the following paragraphs.

As we mentioned, a database is a collection of tables. Each table is similar to a spreadsheet table in which each row is called a *record* and each column is called a *field*. For example, if we need to create a table that contains students' information, we might have the following fields

St_ID	St_Name	St_DateOfBirth	St_Email

Data can be entered to this table so you can get the following table

St_ID	St_Name	St_DateOfBirth	St_Email
45215	John Smith	21/5/1995	jsmith@ox.ac.uk
45287	Alison Green	5/11/1994	agreen@ox.ac.uk
48652	Thomas Li	18/7/1998	tli@ox.ac.uk
51420	Susan Bailey	14/1/1991	sbailey@ox.ac.uk
52201	Will King	3/3/1997	wking@ox.ac.uk

Although this table contains students' information, it does not contain each student's grades. This is fine because the grades have to appear in a different table to reduce data redundancy. This is called *database normalisation*. The grades table might look like

Grade_ID	St_ID	Course_ID	Grade_Value	Comments

Notice how the Grades table is linked to the Students table via `St_ID` which appears in both tables. The field `St_ID` in the Students table is acting as the *primary key* which is a unique id to identify each record in the table. The field `St_ID` in the Grades table is called the *foreign key* and it links to a primary key in a different table. You might have noticed that there is a field called `Course_ID` in the Grades table which is another foreign key to identify a grade's course. This means that there must be another table that contains data for different courses.

Form the previous simple example you should now have an idea of what we mean by a database. It is important to understand the following concepts: database, table, record, field, primary key, foreign key and data normalisation. Next sections will build on this and focus on SQL and how to use it to build a complete database using MySQL.

1.2. What is SQL?

Structured Query Language (SQL) is a relational database language which allows you to create, delete, access and manipulate databases. The following is a list of the main operations that can be formulated with SQL:

- creating new databases
- deleting a database
- creating new tables in a database
- deleting tables from a database
- creating and removing users (database access control)
- executing queries against a database
    - retrieving data from a database
    - inserting records in a database
    - updating records in a database
    - deleting records from a database
- creating stored procedures in a database
- setting permissions on tables and procedures
- creating relationships between tables

1.3. MySQL

MySQL is a *Relational Database Management System* ("RDBMS"). It is used by most modern websites and web-based services as a convenient and fast-access storage and retrieval solution for large volumes of data. A simple example of items which might be stored in a MySQL database would be a site-registered user's name with associated password (encrypted for security), the user registration date, and number of times visited, etc.

MySQL can also be accessed using many tools. It can be easily communicated with via **PHP** (*PHP Hypertext Preprocessor*), a scripting language whose primary focus is to manipulate HTML for a webpage on the server before it is delivered to a client's machine. A user can submit *queries* to a database via PHP, allowing insertion, retrieval and manipulation of information into/from the database.

2 Installation Guide to use MySQL

MySQL can be downloaded from http://dev.mysql.com/downloads/. There are also several MySQL management tools which can be downloaded and installed to allow the manipulation of MySQL. These tools mainly provide an interface to operate on MySQL. Many of these tools are free and provide an easy configuration of MySQL with PHP, e.g., XAMPP, WampServer, AMPPS . Another free MySQL management system is MySQL workbench. It provides database administrators and developers an integrated environment for database design and modelling, SQL development, database administration, database migration. In this course we will be using XAMPP because it is straightforward to install and use.

2.1. XAMPP

XAMPP is a freely available software package which integrates distributions for Apache web server, MySQL, PHP and Perl into one easy installation. If you wish to set up a web server on your home computer, this is the recommended route. We will be using XAMPP for the purposes of this course. The teacher will guide through the process of installing XAMPP in the class.

2.2. phpMyAdmin

Also included within XAMPP is *phpMyAdmin*, a web-based *frontend* ("graphical interface") for MySQL, allowing queries to be submitted via mouse clicks in a web browser or by writing these queries in the SQL box inside phpMyAdmin. Figure 1 shows the main page of phpMyAdmin. In the figure you can see the main tabs which are arranged horizontally at the top.

We will use phpMyAdmin to verify the results of completed examples during this short-course. When directed to do the exercises *in phpMyAdmin*, you should open your web browser and visit the following URL:

http://localhost/phpmyadmin

FYI, most web hosting companies provide a web hosting control panel called *cPanel*. cPanel provides a graphical interface and automation tools to simplify web hosting for customers. cPanel has phpMyAdmin integrated to its system. cPanel has loads of features like website builders, easy transfer of websites, email setup, remote access, etc. For more information see www.cpanel.net.

phpMyAdmin allows the user to write SQL command from the SQL tab. It also provides a mechanism to import SQL command from a file. However, its interface provides other ways to perform tasks using a graphical user interface (GUI). For instance, you can write a command to create a table in your database. You can also achieve this from phpMyAdmin GUI. This course focuses on how to write command-line SQL but I encourage you to explore phpMyAdmin interface.

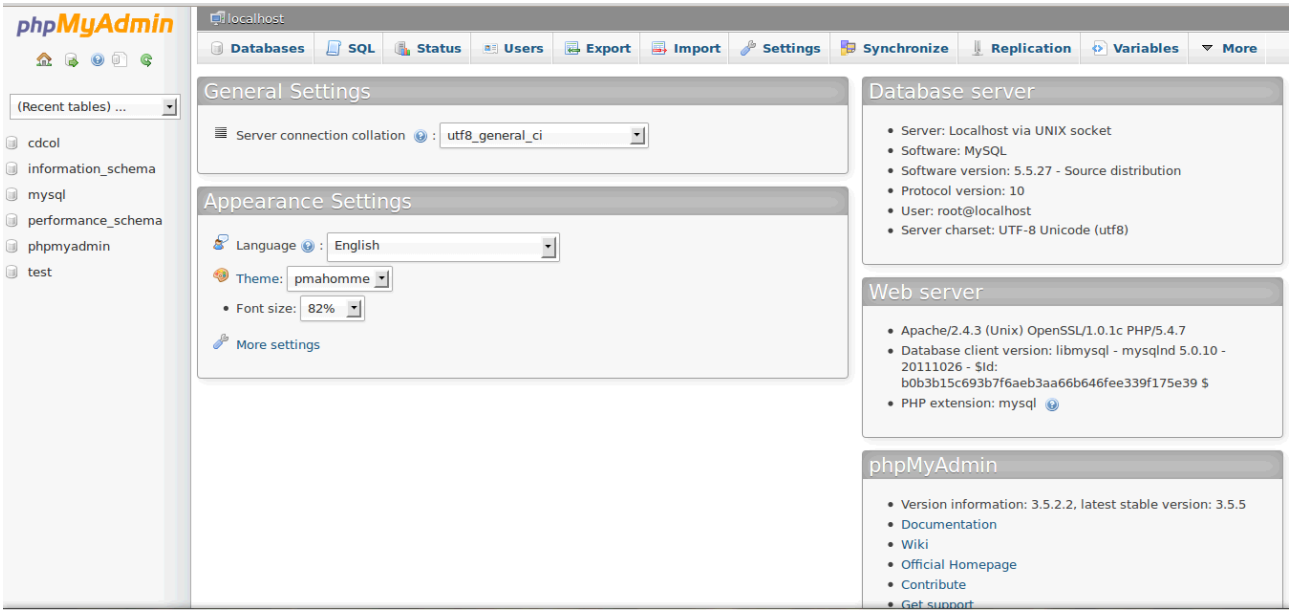


Figure 1. phpMyAdmin main page.

2.3. How to Complete the Exercises

After Installing XAMPP, you are good to go. You need to open XAMPP control panel (usually in C:/xampp/) and start MySQL. The database in the exercises which you are going to practice today is the same database used in other database courses at the IT Learning Programme. The database is for a surgery called St. Giles Surgery. This database contains 4 tables to hold patients, doctors, receptionists and appointments data. Figure 2 shows a schematic diagram of the database. The figure also shows table names (tblPatient, tblDoctors, tblReceptionist and tblAppointment) and field names (or columns) in each table. It also shows the data type for each field (for more information, see section 4.1). The links in the figure reflect the primary-foreign key relationships.

The first few exercises will show how to use phpMyAdmin to write an SQL statement and how to use its GUI instead.

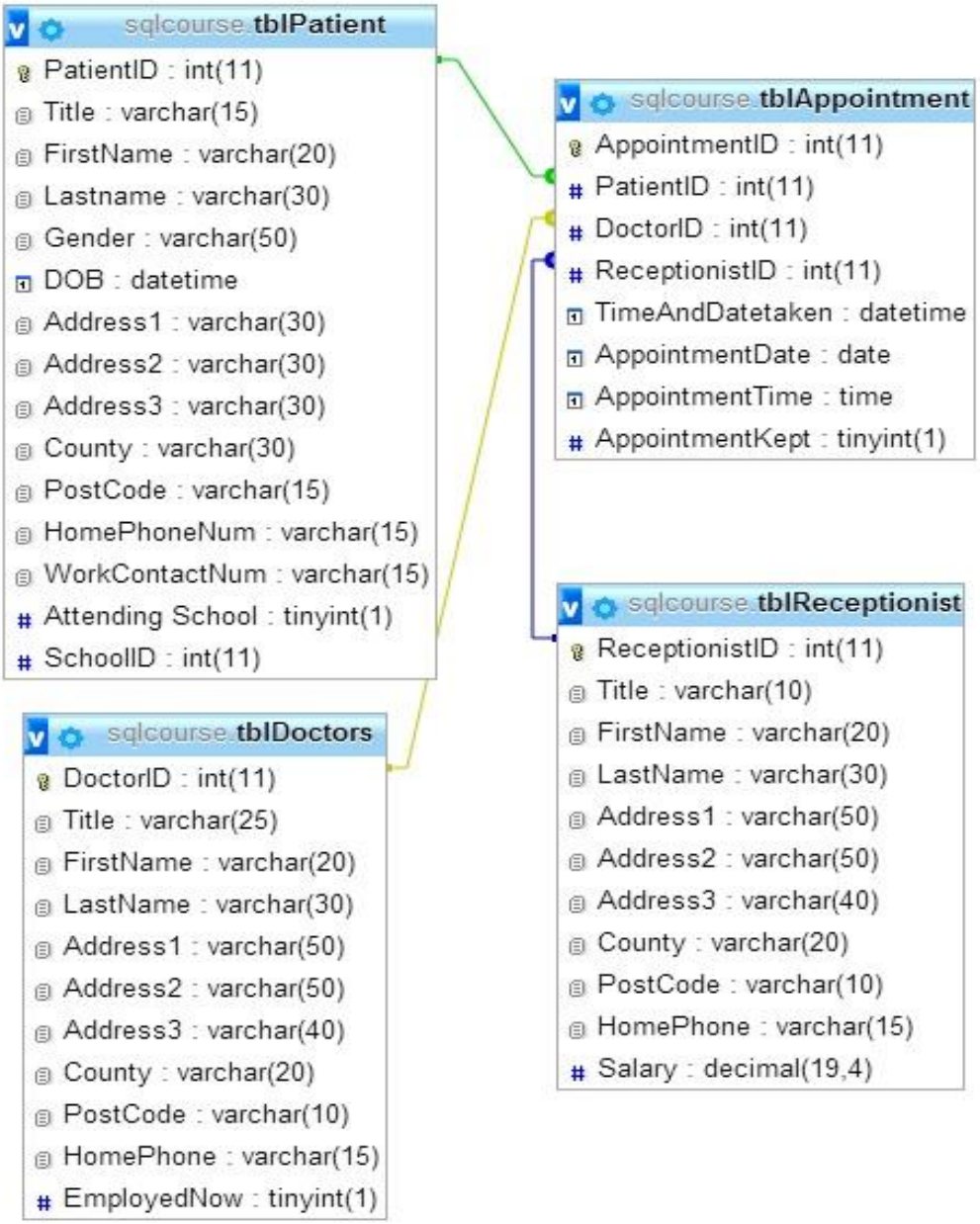


Figure 2. St. Giles Surgery database structure.

3 Setting up MySQL

For the purpose of today's course, you need to have XAMPP installed and running. You then need to do administrative tasks like creating username and password, granting or revoking permissions, creating a database, etc.

3.1. Creating/Editing/Dropping SQL users

It is important to control which users are allowed to connect to the MySQL server and what permissions they are given on what databases. By default, MySQL (within XAMPP) comes with a "*root*" admin user with no password. You should set a password for the root user. You should also create and use a different user with possibly limited permissions to what is needed by the user. For instance, it is not wise to use the root user to connect to a MySQL database from a PHP code. It is better if you connect to it using a different user. You can create MySQL users using phpMyAdmin by clicking on the **Users** tab then click on **Add user**. Enter user name, password and repeated password. You also need to specify that the host is local for the purpose of this course. At the bottom of the page you will find all the permissions which can be granted/revoked from a user. After choosing the required credentials, click on the **Add user** button. phpMyAdmin allows you to drop or edit a user. You can find how to do these from the **Users** tab.

You can also add or delete a user using SQL. The syntax is:

```
CREATE USER user_specification [, user_specification] ...

DROP USER user_name [, user_name] ...
```

For instance, the following two statements add a local MySQL user "sqluser" with a password "test". The second statement drops the user.

```
CREATE USER 'sqluser'@'localhost' IDENTIFIED BY 'test';
DROP USER 'sqluser'@'localhost';
```

There are several other statements in MySQL which allows other user management functionalities. For more information check `GRANT`, `REVOKE`, `RENAME` and `SET PASSWORD` statements in `http://dev.mysql.com/doc/#manual`.

NOTE: In any syntax given in this book, we use [.] to refer to an optional part of a statement. For example, in the statement above (`Drop USER user_name [, user_name]`), the part `[, user_name]` is optional and it can be omitted. If you include one of the optional clauses in a statement, do not type the [square bracket] symbols.

Exercise 1 Create MySQL users	
Suggested time to spend on this exercise is 7 minutes	
NOTE: Make sure that XAMPP/MySQL is running to be able to complete any exercise.	
Task 1 Run XAMPP	**Step 1** After installing XAMPP, open C:/XAMPP. Then open XAMPP control panel. Click **start** MySQL from XAMPP control panel.
	Note that you can stop or configure MySQL from the same control panel.
Task 2 Start phpMyAdmin and familiarise yourself with it	**Step 1** Open any browser (Chrome is preferred) then type the following in the address bar `Localhost/phpmyadmin` **Step 2** phpMyAdmin has many tabs. Please check these tabs to become familiar with its interface. Also, notice the tree view on the left hand side of phpMyAdmin page. This view allows you to access databases/tables faster.
Task 3 Create a MySQL user using phpMyAdmin interface	**Step 1** Click on the **Users** tab, click on **Add user**. Enter user name (**sqluser1**), password (**test**) and repeated password (**test**). From Host, select **Local**. **Step 2** In the "Global privileges" panel, **Check all** the permissions to create an admin user. Then click the **Go** button.
Task 4 Alternatively, you can create a user by writing a SQL statement.	**Step 1** Click on the **SQL** tab. In the empty box type the following `CREATE USER 'sqluser2'@'localhost' IDENTIFIED BY 'test';` **Step 2** Click **Go**.

3.2. Creating/Dropping Databases

Using phpMyAdmin, you now need to create a new database. To do that, you need to click on the **Databases** tab and enter a database name then click **Create**. phpMyAdmin allows the user to delete a database from its interface as well.

Alternatively, you can write a SQL code to create/drop a database instead. The syntax is:

```
CREATE DATABASE [IF NOT EXISTS] db_name
DROP DATABASE [IF EXISTS] db_name
```

Note that to connect to a MySQL database from PHP you need to specify a username, password and database name.

Exercise 2 Create MySQL database	
Suggested time to spend on this exercise is 4 minutes	
Task 1 Create a database using phpMyAdmin interface	**Step 1** From the main window of phpMyAdmin, click on the Databases tab. **Step 2** Enter a database name (sqlcourse). **Step 3** Click Create.
Task 2 Alternatively, you can create a database using a SQL statement.	**Step 1** Click on the SQL tab. In the empty box type the following `CREATE DATABASE IF NOT EXISTS sqlcourse;` **Step 2** Click Go.
Notes • The statement in Task 2 will do nothing as you have already created a database with this name. • Notice on the left side of phpMyAdmin that the database name appeared. • In fact, you can access databases from the left side view (tree view) of phpMyAdmin. You can later access all created tables within each database.	

4 Creating Tables

The `CREATE TABLE` statement is used to construct new tables, in which rows of data can be stored. Its general syntax is

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (col_name column_definition,...)
    [table_options]
    [partition_options]
```

The ***column_definition*** is the description of a column in the table. The general format of the column definition is:

```
column_definition:
    data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
      [COMMENT 'string']
      [COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}]
      [STORAGE {DISK|MEMORY|DEFAULT}]
      [reference_definition]
```

We will be discussing some of these options in the following examples. The most important definition is the ***data_type*** which is described in Section 4.1.

4.1. Data Types

SQL usually supports a number of data types in several categories: numeric types, date and time types, and string (character and byte) types. The most common ones are:

```
   INTEGER[(length)] [UNSIGNED] [ZEROFILL]
   FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
   DATE
   TIME
   CHAR[(length)] [CHARACTER SET charset_name] [COLLATE collation_name]
   BINARY[(length)]
   TEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
```

MySQL supports much more data types which are variations of the common data types. Here is a list of these data types in MySQL:

```
   BIT[(length)]
   TINYINT[(length)] [UNSIGNED] [ZEROFILL]
   SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
   MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
```

```
INT[(length)] [UNSIGNED] [ZEROFILL]
INTEGER[(length)] [UNSIGNED] [ZEROFILL]
BIGINT[(length)] [UNSIGNED] [ZEROFILL]
REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
DATE
TIME
TIMESTAMP
DATETIME
YEAR
CHAR[(length)]
   [CHARACTER SET charset_name] [COLLATE collation_name]
VARCHAR(length)
   [CHARACTER SET charset_name] [COLLATE collation_name]
BINARY[(length)]
VARBINARY(length)
TINYBLOB
BLOB
MEDIUMBLOB
LONGBLOB
TINYTEXT [BINARY]
   [CHARACTER SET charset_name] [COLLATE collation_name]
TEXT [BINARY]
   [CHARACTER SET charset_name] [COLLATE collation_name]
MEDIUMTEXT [BINARY]
   [CHARACTER SET charset_name] [COLLATE collation_name]
LONGTEXT [BINARY]
   [CHARACTER SET charset_name] [COLLATE collation_name]
ENUM(value1,value2,value3,...)
   [CHARACTER SET charset_name] [COLLATE collation_name]
SET(value1,value2,value3,...)
   [CHARACTER SET charset_name] [COLLATE collation_name]
```

4.2. Primary Key

The *PRIMARY KEY* constraint uniquely identifies each record in a database table. It is important to distinguish records in a table. For instance, student ID is used as a unique key for each student in a school. Primary key values must be unique and cannot be NULL.

Exercise 3 Create MySQL table	
Suggested time to spend on this exercise is 10 minutes	

Task 1
Create table **tblDoctors**
	Step 1
	Click on the sqlcourse database. You can find it on left side panel that contains all databases. This step is important in every exercise. Without clicking on the database the next step won't work.
	Step 2
	Alternatively, you can write the following statement instead of clicking on the database to tell SQL that you want to use a specific database.
	`USE sqlcourse;`
	Step 3
	Click on the SQL tab. In the empty box type the following
	`CREATE TABLE tblDoctors (`
	`DoctorID int(11) NOT NULL AUTO_INCREMENT,`
	`Title varchar(25) DEFAULT 'Dr',`
	`FirstName varchar(20) DEFAULT NULL,`
	`LastName varchar(30) DEFAULT NULL,`
	`Address1 varchar(50) DEFAULT NULL,`
	`Address2 varchar(50) DEFAULT NULL,`
	`Address3 varchar(40) DEFAULT NULL,`
	`County varchar(20) DEFAULT NULL,`
	`PostCode varchar(10) DEFAULT NULL,`
	`HomePhone varchar(15) DEFAULT NULL,`
	`EmployedNow tinyint(1) DEFAULT 0,`
	`PRIMARY KEY (DoctorID),`
	`KEY DoctorID (DoctorID),`
	`KEY PostCode (PostCode)`
	`)ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;`
	Step 4
	Click Go.
	Step 5
	Correct any coding errors that become evident.

Explanation:

The above statement creates a table called **tblDoctors** with several fields. Please note the following:

- `DEDAULT` is used to give a default value for the field when entering a new record.
- `NULL` means that the field can be empty. `NOT NULL` means the opposite.
- `PRIMARY KEY` is used to specify the field name which is to be used as a primary key.
- `KEY` is normally a synonym for INDEX. This is usually used to identify fields in a table which can be linked to primary keys in other tables. Check Exercise 5.
- `ENGINE=InnoDB`: specifies the MySQL database engine as there are several MySQL engines. It is out of the scope of this course to describe these engines.
- `AUTO_INCREMENT`: specifies which field is an auto-generated number.
- `AUTO_INCREMENT=1`: the first number to start with.
- `CHARSET` is a synonym for `CHARACTER SET`. MySQL allows storing data using a variety of character sets and to perform comparisons according to a variety of collations. For more information search for "MySQL charset".
- Not shown in this example – `UNIQUE`: creates a constraint such that all values in the field must be distinct. However, in most MySQL engines, unique fields can be null. This makes it different from a primary key.

Exercise 4 Create the other 3 MySQL tables using the import facility.

So we have the SQL statements already written in a text file called tables.sql. We will import the ready-made SQL statements to create the remaining 3 tables for the purpose of today's course. We are doing this way because I don't want you to keep on typing these statements since we have a limited time to finish the course. So we will practice the import functionality in phpMyAdmin which allows importing SQL statements written on a file. For more information check Chapter 8.

Suggested time to spend on this exercise is 5 minutes

Task 1	**Step 1**
Import an existing SQL file to MySQL database.	Make sure you are within the sqlcourse database. **Step 2** NOTE: To edit the file before importing it to MySQL, you can open it using Notepad or any similar text editor. Please let me know if you have any problem. Click on the Import tab and click on the Choose File button. Locate the file **tables.sql** and click Open. **Step 3** Click the Go button.

Note: If you are interested to write the SQL statements instead of importing them, the **tables.sql** file contains the following `CREATE` statements. In addition, you should be able to see the created tables in phpMyAdmin.

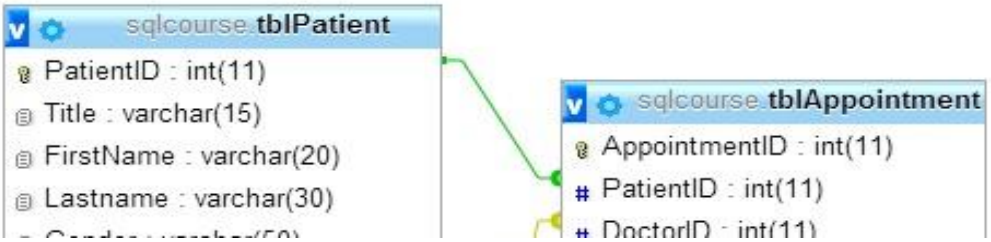
```
# First table tblReceptionist. By the way, this is a comment in MySQL.
CREATE TABLE tblReceptionist (
  ReceptionistID int(11) NOT NULL AUTO_INCREMENT,
  Title varchar(10) DEFAULT 'Mrs',
  FirstName varchar(20) DEFAULT NULL,
  LastName varchar(30) DEFAULT NULL,
  Address1 varchar(50) DEFAULT NULL,
  Address2 varchar(50) DEFAULT NULL,
  Address3 varchar(40) DEFAULT NULL,
  County varchar(20) DEFAULT 'Oxfordshire',
  PostCode varchar(10) DEFAULT NULL,
  HomePhone varchar(15) DEFAULT NULL,
  Salary decimal(19,4) DEFAULT '0.0000',
  PRIMARY KEY (ReceptionistID),
  KEY ReceptionistID (ReceptionistID),
  KEY PostCode (PostCode)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=1;

# Second table tblPatient
CREATE TABLE tblPatient (
  PatientID int(11) NOT NULL AUTO_INCREMENT,
  Title varchar(15) DEFAULT NULL,
  FirstName varchar(20) DEFAULT NULL,
  Lastname varchar(30) NOT NULL,
  Gender varchar(50) DEFAULT 'Female',
  DOB datetime DEFAULT NULL,
  Address1 varchar(30) DEFAULT NULL,
  Address2 varchar(30) DEFAULT NULL,
  Address3 varchar(30) DEFAULT NULL,
  County varchar(30) DEFAULT 'Oxfordshire',
  PostCode varchar(15) NOT NULL,
  HomePhoneNum varchar(15) DEFAULT NULL,
  WorkContactNum varchar(15) DEFAULT NULL,
  Attending School tinyint(1) DEFAULT '0',
  SchoolID int(11) DEFAULT NULL,
  PRIMARY KEY (PatientID),
  KEY HomePhoneNum (HomePhoneNum),
  KEY Lastname (Lastname,FirstName),
  KEY PostCode (PostCode),
 KEY SchoolID (SchoolID),
  KEY WorkContactNum (WorkContactNum)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

# Third table tblAppointment
CREATE TABLE tblAppointment (
  AppointmentID int(11) NOT NULL AUTO_INCREMENT,
  PatientID int(11) NOT NULL,
  DoctorID int(11) NOT NULL,
  ReceptionistID int(11) NOT NULL,
  TimeAndDatetaken datetime NOT NULL,
  AppointmentDate date DEFAULT NULL,
  AppointmentTime time DEFAULT NULL,
  AppointmentKept tinyint(1) DEFAULT '0',
  PRIMARY KEY (AppointmentID),
  KEY DoctorID (DoctorID),
  KEY AppointmentID (AppointmentID),
  KEY PatientID (PatientID),
  KEY ReceptionistID (ReceptionistID)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=1;
```

4.3. Linking Tables via Primary – Foreign Keys

A *FOREIGN KEY* is a field in one table that points to a PRIMARY KEY in another table. This constraint is important to link tables together. For instance, `PatientID` field in the `tblAppointment` table is foreign key for the primary key `PatientID` in the `tblPatient` table.

There are different ways to specify a foreign key. As we have already created our tables, we can alter them to add the foreign key constraints. The syntax is:

```
ALTER TABLE table_name ADD [CONSTRAINT [symbol]] FOREIGN KEY (column_name)
REFERENCES the_other_table_name (column_name) [ON DELETE CASCADE] [ON
UPDATE CASCADE]
```



PatientID is a primary key in tblPatient

PatientID is a foreign key in tblAppointment

Exercise 5 Create Foreign Key constraints

This exercise allows you to create a constraint to a table by creating a primary – foreign key constraint.

Suggested time to spend on this exercise is 5 minutes

Task 1	**Step 1**
Add foreign key constraints	Click on the **SQL** tab, type the following statement and then click **Go**: **Note:** you can find this statement written in the **const.sql** file so you can import it instead but it is important to understand the syntax.

```
ALTER TABLE tblAppointment

  ADD CONSTRAINT FOREIGN KEY (DoctorID) REFERENCES tblDoctors (DoctorID)
ON DELETE CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT FOREIGN KEY (PatientID) REFERENCES tblPatient (PatientID)
ON DELETE CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT FOREIGN KEY (ReceptionistID) REFERENCES tblReceptionist
(ReceptionistID) ON DELETE CASCADE ON UPDATE CASCADE;
```

Explanation:

- This is one big statement which adds three constraints to the table `tblAppointment`.
- `CASCADE`: on deleting or updating a row from a parent table (e.g., tblPatient), automatically deletes or updates the matching rows in the child table (e.g., tblAppointment).

5 Manipulating Data in Tables

Once we got the tables set up with fields and links, we need to enter some data in. Data can be inserted directly using SQL statements or imported from a pre-written file. Remember that data from different formats can be imported, e.g., CSV (e.g., search for "Import CSV to MySQL"). In this section, we will be focusing on manipulating data in tables by writing SQL statements.

5.1. INSERT Statement

In SQL, you can use the `INSERT` statement to add rows of data to an existing table. With this statement, you can add new rows or populate a table with rows taken from another table.

The basic syntax is:

```
INSERT  INTO tbl_name (col_name1, col_name2 ...) VALUES (val1, val2 …)
```

The general syntax of the INSERT statement which contains more options is:

```
INSERT [LOW PRIORITY | DELAYED | HIGH PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

Exercise 6 Insert data to tables	
Suggested time to spend on this exercise is 7 minutes	
Task 1 Insert one row to the **tblDoctors** table	**Step 1** Click on the SQL tab. In the empty box type the following and then click Go:
`INSERT INTO tblDoctors (DoctorID, Title, FirstName, LastName, Address1, Address2, Address3, County, PostCode, HomePhone, EmployedNow) VALUES` `(1, 'Dr', 'Joe', 'Blowphelt', '12, Hill St', 'Witney', NULL, 'Oxfordshire', 'OX3 5EW', '34432432', 1);`	
Task 2 Import the remaining data to tables	**Step 1** Open the file **Data.sql** using any text editor e.g., Notepad. Spend some time reading the SQL `INSERT` statements in the file. **Step 2** Import the file **Data.sql** to the sqlcourse database (user the same steps as in Exercise 4).
Task 3 Check data in phpMyAdmin	**Step 1** From the left panel of phpMyAdmin, click on the sqlcourse database. You should be able to see the 4 tables. Explore the fields and records in each table. **Step 2** Click on the Browse tab to see the data. **Step 3** Click on the Structure tab to show information about each field in a table.

5.2. UPDATE Statement

The easiest way to update a value in a table is by viewing the content of a table via phpMyAdmin then editing a field to change a specific value. This works well if one cell is to be edited. If many values are needed to get changed then this becomes tedious. The solution to this is to use the `UPDATE` statement.

With the `UPDATE` statement, you can change one or more values in one or more tables. To achieve this, use the table(s) name to indicate which table needs to be updated and field(s) name to specify which column(s) within the table(s) to update. The `WHERE` clause of an `UPDATE` statement specifies which rows must be changed (see Section 6.2); the `SET` clause assigns new values to one or more columns. The basic and common syntax is

```
UPDATE table_reference SET col_name1={expr1|DEFAULT} [,
col_name2={expr2|DEFAULT}] ...
[WHERE where_condition]
```

The more detailed syntax is:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Exercise 7 Update data in a table	
Suggested time to spend on this exercise is 4 minutes	
Task 1 Update the salary of a receptionist	**Step 1** In the tblReceptionist table, check **Sarah Peters** salary. It should be 9875. **Step 2** You need to change the salary of the receptionist **Sarah Peters** to 10000. To update the salary, write the following SQL statement and click Go:
`UPDATE tblReceptionist SET Salary=10000 WHERE LastName='Peters'`	
Task 2 Check if the salary has changed from the `tblReceptionist` table.	
NOTE: • The `WHERE` statement is covered in Section 6.2. It is used to specify a subset of records in a table. • You can view and update the data using phpMyAdmin interface. However, this can be achieved on one row at a time. SQL statements allow you to update multiple rows in one command. **Optional questions (discuss the answers with the teacher if you want):** • What if you have two or more receptionists of last name Peters? Using the previous update statement, what will happen? • What happens if we drop the WHERE part from the update statement?	

5.3. DELETE Statement

The DELETE statement removes rows from a table. A basic syntax for the DELETE statement is:

```
DELETE FROM tbl_name [WHERE where_condition]
```

A more complete syntax with more options:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Exercise 8 Delete data from a table	
Suggested time to spend on this exercise is 5 minutes	
Task 1 Before we delete anything, let's insert a new receptionist to the **tblReceptionist**	**Step 1** Insert a receptionist. Table **tblReceptionist** have several fields. You just need to insert a few fields. The following are the only fields to enter (keep the remaining fields empty): ReceptionistID: 6 FirstName: Sam LastName: Lee Refer to Exercise 6 for more information on how to insert data. **Step 2** Check to see if data was entered correctly.
Task 2 Delete the record for the recently entered receptionist	**Step 1** Write the following `DELETE FROM tblReceptionist WHERE ReceptionistID=6` **Step 2** Check whether data was deleted.

6 Queries

6.1. SELECT Statement

The SELECT statement is used to query data from tables. The retrieved rows are selected from one or more table. Such a result table can be used as the basis of a report, for example.

The basic syntax of the SELECT statement is:

```
SELECT select_expr [, select_expr ...] FROM table_name
```

Each ***select_expr*** indicates a column that you want to retrieve. * is used instead of ***select_expr*** as a wildcard if you want to retrieve all columns from a table.

The complete syntax of the SELECT statement is:

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_name
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name' export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

In general, clauses used must be given in exactly the order shown in the syntax description. For example, a HAVING clause must come after any GROUP BY clause and before any ORDER BY clause.

Exercise 9 Querying data from a table	
Suggested time to spend on this exercise is 4 minutes	
Task 1 Retrieve all doctors' information	**Step 1** This requires writing a `SELECT` statement to retrieve all columns from the **tblDoctors** table. To do that write the following statement in the SQL box and click **Go**.
`SELECT * FROM tblDoctors`	
Task 2 Retrieve all receptionists' first and last names	**Step 1** Note that there is a link called **Show query box** usually at the top of the page which you click to keep the recent query you wrote. Click on it and write the following
`SELECT FirstName, LastName FROM tblReceptionist`	
Task 3 [Optional] You might want to practice other queries yourself. For example, write a query to retrieve all patients first and last names.	
Note: When you write a query in the SQL box in phpMyAdmin and run it, you can see the total number of retrieved records on the top of the page. A text in a green box should read for example "Showing rows 0 - 29 (86 total, Query took 0.0010 sec)". In this example the query returned 86 records, the first 30 (0-29) are displayed. You can show all records if you click on the **Show all** button or you can use the arrows next to the **Show all** button to browse the remaining records. The time needed to execute the query is also displayed here, e.g., 0.0010 sec in this example.	

6.2. Where Clause

In the `WHERE` clause, a condition is used to select rows from a table. These selected rows form the intermediate result of the `WHERE` clause. The `WHERE` clause acts as a kind of filter.

Its syntax is

```
SELECT select_expr [, select_expr ...] FROM table [WHERE where_condition]
```

Exercise 10 Querying specific records	
Suggested time to spend on this exercise is 5 minutes	
Task 1 Retrieve the names of all the female patients	**Step 1** To do that write the following statement in the SQL box and click **Go**.
`SELECT Title, FirstName, LastName FROM tblPatient WHERE Gender='Female'`	
Task 2 [Optional tasks] You might **want** to practice other queries yourself. For example, - write a query to retrieve all receptionists who live in Summertown. - write a query to retrieve receptionist details whose salary equals £9400. - write a query to retrieve appointments made on '2013-07-02'	
Note: When comparing two values (e.g., `Gender='Female'` in the previous example), we used two single quotations around the work Female because it is text. Remember that text and dates have to appear between two single quotes. Numbers do not need to appear between quotes.	

6.3. Comparisons and Conditions

We set conditions within the `WHERE` clause. The condition could be an expression, for example, `83` or `15 * 100` as already discussed. Alternatively, it could be a comparison or relation operator with another value, for example `<83` or `>=100`).

Its syntax is

```
WHERE column_name operator expression_value
```

The value of the "column_name" is compared with the value of the expression. The result will be true, false, or unknown. SQL supports the comparison operators shown in Table 1. Multiple conditions can be combined using the logical operators shown in Table 2. For example we can write the following as a condition:

```
WHERE column_name1 operator value1 AND column_name2 operator value2
```

Table 1. Comparison operators

Comparison Operator	Meaning
=	Equal to (as in Exercise 10)
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

Table 2. Logical operators

Name	Description
`AND, &&`	Logical AND
`NOT, !`	Negates value
`||, OR`	Logical OR
`XOR`	Logical XOR

Date and time comparisons. Please note that to compare date or time, you need to specify the date or time in two single quotes as you do for strings. Use the same format as specified in the database. For instance, if the date is saved in a database in YYYY-DD-MM format (which represents 4 digits year – two digits day – two digits month), then you need to compare this date using the same format, as for example:

```
WHERE column_name operator '1995-25-07'
```

Exercise 11 Querying data – using conditions	
Suggested time to spend on this exercise is 7 minutes	
Task 1 Retrieve the names, phones and salaries of receptionists whose salary is more than 10000	**Step 1** To do that write the following statement in the SQL box and click **Go**.
`SELECT FirstName, LastName, HomePhone, Salary FROM tblReceptionist WHERE Salary>10000`	
Task 2 Retrieve all male patients who live in Oxford	**Step 1** Write the following and click **Go**
`SELECT * FROM tblPatient WHERE Gender='Male' AND Address3='Oxford'`	
Task 3 [Optional] You might want to practice other queries yourself. For example: - change the greater than sign in Task 1 to greater than or equal. - write a query to retrieve all PatientIDs who were born after 1/1/1988. - retrieve receptionists who get salary between 9000 and 12000.	

7 Advanced Queries

7.1. Sorting Data – ORDER BY Clause

To sort the result of a query, MySQL uses an `ORDER BY` clause. The syntax for the `ORDER BY` clause within a `SELECT` statement is as follows:

```
[ORDER BY {col_name | position} [ASC | DESC]
```

Columns selected for output can be referred to in `ORDER BY` and `GROUP BY` (see Section **Error! Reference source not found.**) clauses using column names, olumn aliases, or column positions. Column positions are integers and begin with 1. To sort in reverse order, add the `DESC` (descending) keyword to the name of the column in the `ORDER BY` clause that you are sorting by. The default is ascending order; this can be specified explicitly using the `ASC` keyword.

Exercise 12 Retrieving sorted records	
Suggested time to spend on this exercise is 5 minutes	
Task 1 Retrieve all appointments sorted by appointment date. Try ascending order first then change it to descending order.	**Step 1** To do that write one of the following statements in the SQL box and click **Go**. The first two statements are the same and they sort the data in ascending order while the third statement sorts the data in descending order.

`SELECT * FROM tblAppointment ORDER BY AppointmentDate`
`SELECT * FROM tblAppointment ORDER BY AppointmentDate ASC`
`SELECT * FROM tblAppointment ORDER BY AppointmentDate DESC`	

Task 2 Retrieve all appointments taken after 1/7/2013 and sorted by appointment date in descending order.	**Step 1** Write the following
`SELECT * FROM tblAppointment WHERE TimeAndDatetaken>'2013-07-01' ORDER BY AppointmentDate DESC`	

7.2. Querying Multiple Tables

You can query different columns from multiple tables. There are different ways to do this. However, one has to be careful when retrieving data from multiple tables as unwanted records might be retrieved. The straightforward way to retrieve rows from multiple tables is by query two or more tables in one `SELECT` statement. For instance, you can get all records from two tables a follows

```
SELECT * FROM tbl_name_1, tbl_name_2 [WHERE where_condition]
```

7.3. Pseudonyms for Table or Column Names

When multiple table specifications appear in the `FROM` clause, it is sometimes easier to use so-called pseudonyms. Another name for pseudonym is an alias. Pseudonyms are temporary alternative names for table names. This helps distinguish between fields in multiple tables. Pseudonyms can also be used to give an alias name for a column (field). The syntax is:

```
SELECT alias1.*, alias2.fieldX FROM tbl_name_1 AS alias1, tbl_name_2 AS alias2
[WHERE where_condition]
```

If a field name exists in two tables with the same name, we can use a pseudonym to distinguish between the two fields. For instance, if `table1` and `table2` both have a field called `fieldX`, we can write the following to retrieve `fieldX` from table1

```
SELECT t1.fieldX FROM table1 AS t1, table2 AS t2 WHERE t1.fieldX = t2.fieldX
```

Exercise 13 Querying multiple tables	
Suggested time to spend on this exercise is 7 minutes	
Task 1 Retrieve receptionist names and appointment dates from the receptionist and appointment tables, labelling the tables as "r" and "app" respectively.	**Step 1** To do that write the following statement in the SQL box and click **Go**. **Step 2** How many records have been retrieved?
`SELECT r.FirstName, r.LastName, app.AppointmentDate FROM tblReceptionist AS r, tblAppointment as app`	
Task 2 Add the doctor last name to the query in the previous task	**Step 1** Write the following. Please notice the difference between the two queries.
`SELECT r.FirstName, r.LastName, d.LastName, app.AppointmentDate FROM tblReceptionist AS r, tblAppointment as app, tblDoctors AS d`	
Question: How many records have been returned in Task 1 and Task 2? Why?	
Note: You might have noticed that the total number of rows retrieved using the query in Task 1 is number of records in `tblReceptionist` multiplied by the number of records in `tblAppointment`. What do you think about the number of records in Task 2? Apparently, what happened is that each record from the first table was repeated with each record in the second table. This is called Cartesian product. In fact, we did not want this to happen. To avoid this we need to utilise the primary-foreign keys relationship. Check out the next exercise.	

Exercise 14 Querying multiple tables: use primary-foreign keys relationship	
Suggested time to spend on this exercise is 4 minutes	
Task 1 Retrieve the receptionist names and date of the appointments they made	**Step 1** To do that write the following statement in the SQL box and click **Go**.
`SELECT r.FirstName, r.LastName, app.AppointmentDate FROM tblReceptionist AS r, tblAppointment as app WHERE app.ReceptionistID = r.ReceptionistID`	

7.4. Subquery (inner SELECT)

A subquery is a `SELECT` statement within another statement. In other words, a table expression can be called from within another table expression. The called table expression is a subquery. The syntax for it is

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2 [WHERE …])
```

Exercise 15 Querying multiple tables: use primary-foreign keys relationship	
Suggested time to spend on this exercise is 5 minutes	
Task 1 Retrieve appointments created by receptionist Mrs Burns	**Step 1** To do that write the following statement in the SQL box and click **Go**.
`SELECT * FROM tblAppointment WHERE ReceptionistID = (SELECT ReceptionistID from tblReceptionist WHERE LastName = 'Burns')`	
Task 2 [Optional Task] Retrieve all appointment dates for Dr Down.	

Since we used an equal sign for the inner subquery so far, this means that the inner query should only return one record. If the inner subquery returns more than one record, MySQL will issue an error message saying so. However, sometimes we do not know if the inner subquery will return more than one record or even no records. To avoid this situation, check the next section (`IN` operator).

7.5. The IN Operator

The use of the `IN` operator in a `SELECT` statement makes multiple comparisons easier. The condition with the `IN` operator has two forms. The first form is when comparing a field with a list of values separated by commas. For instance, if you use the equal sign to compare a column to multiple values, it gives a long statement as follows:

```
SELECT * FROM t1 WHERE column1 = 'value1' OR column1 = 'value2' OR
       column1 = 'value3' …
```

Instead, use the `IN` operator to make the `SELECT` statement shorter and easier to read:

```
SELECT * FROM t1 WHERE column1 IN ('value1','value2','value3' …)
```

The second form is to use the `IN` operator with a subquery. This happens when a value of a field from a one table is to match one or more from another table. Try the following exercise.

Exercise 16 Querying multiple tables: use IN operator	
Suggested time to spend on this exercise is 7 minutes	
Task 1 Retrieve appointments created by receptionists who get salary greater than 10000.	**Step 1** Try the following SQL statement.
`SELECT * FROM tblAppointment WHERE ReceptionistID = (SELECT ReceptionistID from tblReceptionist WHERE Salary >10000)`	
	Step 2 Change the = sign in the previous statement to the `IN` operator. Why do you think we needed the `IN` operator here instead of the equal sign?
Task 2 [Optional Task] Retrieve patient records that have appointments with Dr Down.	**Step 1** Try the following SQL then change the first = sign to the `IN` and then change the second = sign to `IN`. Discuss this with the teacher if you have any question.
`SELECT * FROM tblPatient WHERE PatientID = (SELECT PatientID FROM tblAppointment WHERE DoctorID = (SELECT DoctorID FROM tblDoctors WHERE LastName='Down'))`	
Note You might have noticed that you only needed to change the first equal sign in the last query to get correct result. Why? Also, try to change `LastName='Down'` to `County='Oxfordshire'`. Do you need to change the second equal sign to `IN` this time?	

7.6. Basic String Comparison Functions

MySQL has functions for comparing strings of text (alphanumeric values). The basic ones are `LIKE`, `NOT LIKE` and `STRCMP()`. The `LIKE` operator is used to select

alphanumeric values with a particular pattern or mask. `NOT LIKE` is the negation of `LIKE`.

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities. You to use '_' to match any single character and '%' to match an arbitrary number of characters (including zero characters).

`STRCMP()` is used to compare two strings. It returns 0 if both strings are the same, it returns -1 when the first string is smaller than the second according to the defined order and 1 when second string is smaller the first one.

The standard syntax for these operators is:

```
Str1 LIKE Str2
Str1 NOT LIKE Str2
STRCMP (expr1, expr2)
```

Exercise 17 The use of LIKE and NOT LIKE	
Suggested time to spend on this exercise is 5 minutes	
Task 1 Retrieve any receptionist(s) who live in OX4 postcode area	**Step 1** To do that write the following statement in the SQL box and click **Go**.
`SELECT * FROM tblReceptionist WHERE PostCode LIKE 'OX4%'`	
Task 2 [Optional Tasks] - Try to retrieve the receptionist(s) who do not live in OX4 postcode area. - Find all patients that have last name ending with the letter 's' - Find all patients that have first name of 4 characters length only. Hint: use the underscore matching pattern.	

Exercise 18 The use of STRCMP()	
Suggested time to spend on this exercise is 3 minutes	
Task 1 Retrieve all patients whose last name starts with a letter from the range P – Z, arranged by last name in ascending order.	**Step 1** To do that write the following statement in the SQL box and click **Go**.
`SELECT * FROM tblPatient WHERE STRCMP(LastName,'P')=1 ORDER BY LastName`	

7.7. The BETWEEN Operator

SQL supports a special operator that determines whether a value occurs within a given range of values. Its basic syntax is

```
WHERE column_name BETWEEN value1 AND value2
```

Exercise 19 Querying data – BETWEEN operator	
Suggested time to spend on this exercise is 5 minutes	
Task 1 Retrieve all receptionists who get salary between 9000 and 12000.	**Step 1** To do that write the following statement in the SQL box and click **Go**.
`SELECT * FROM tblReceptionist WHERE Salary BETWEEN 9000 AND 12000`	
Optional Tasks • Retrieve all appointments between 2/7/2013 and 4/7/2013 • Retrieve all appointments for Dr Blowphelt between 2/7/2013 and 4/7/2013 sorted by appointment date	

8 Importing and Exporting

MySQL allows creating a dump file from a database, or restoring data from a file to a live database. This process allows importing and exporting database structures (e.g., tables) and data (e.g., the content of tables). It is possible to move information between different MySQL databases or even between MySQL and different SQL databases like MS Access, Oracle, SQL Server, etc.

8.1. Migration from/to MySQL Database only

MySQL allows the user to export a database and dump it as a ".sql" file which contains all the SQL statements needed to create the database structure and all the SQL statements needed to insert data into these tables. The export also generates other necessary statements which for instance are used to create a database, users, etc.

To export a database from phpMyAdmin, click on the Export tab. Several file formats can be used during export. The SQL file format is the most common one to use when exporting data from one place to another. Notice that phpMyAdmin allows the user to customise the exportation to their needs. For instance, it is possible to export one table instead of the whole database. Also, notice that phpMyAdmin allows exporting data in several file formats including SQL, CSV, PDF, etc.

Importing databases to MySQL is also possible. To do that in phpMyAdmin, use the Import tab. phpMyAdmin allows several file format for importing data to MySQL. Again, ".sql" is the common one. Finally, other MySQL administration tools like workbench have similar way to export or import databases to MySQL.

9 What is Next?

Now that you have completed this short-course, it is hoped you have a better grasp of the fundamental principles of MySQL. So what should you do next?

Attend the MySQL Further Techniques course

ITLP offer another course on MySQL which contains more advanced concepts. For more information ask the teacher or course administrator.

Explore phpMyAdmin

phpMyAdmin contains much more things than what we covered in this book. For instance, have a look on Triggers, Designer, Events, etc. For more information about it please visit

http://docs.phpmyadmin.net/en/latest/
http://www.siteground.com/tutorials/phpmyadmin/

Read a book or tutorials about MySQL or SQL in general

A detailed illustration about MySQL can be found on:

http://dev.mysql.com/doc/
http://www.tutorialspoint.com/mysql/
http://www.w3schools.com/SQL/

There are many other websites that has nice tutorial about MySQL. Just Google it.

Thank you for attending this short-course, and **Good Luck!**
