# Data Management: Databases - MySQL Further Techniques for Researchers

iT Learning Centre

iT services

University of Oxford

# How to Use this User Guide

This handbook accompanies the taught sessions for the course. Each section contains a brief overview of a topic for your reference and then one or more exercises.

Exercises are arranged as follows:

- A title and brief overview of the tasks to be carried out;
- A numbered set of tasks, together with a brief description of each;
- A numbered set of detailed steps that will achieve each task.

Some exercises, particularly those within the same section, assume that you have completed earlier exercises. Your teacher will direct you to the location of files that are needed for the exercises. If you have any problems with the text or the exercises, please ask the teacher or one of the demonstrators for help.

This book includes plenty of exercise activities – more than can usually be completed during the hands-on sessions of the course. You should select some to try during the course, while the teacher and demonstrator(s) are around to guide you. Later, you may attend follow-up sessions at IT Learning Centre (ITLC) called Computer8, where you can continue work on the exercises, with some support from IT teachers. Other exercises are for you to try on your own, as a reminder or an extension of the work done during the course.

## Text Conventions

A number of conventions are used to help you to be clear about what you need to do in each step of a task.

- In general, the word **press** indicates you need to press a key on the keyboard. **Click**, **choose** or **select** refer to using the mouse and clicking on items on the screen. If you have more than one mouse button, click usually refers to the left button unless stated otherwise.
- Names of keys on the keyboard, for example the Enter (or Return) key are shown like this ENTER.
- Multiple key names linked by a + (for example, CTRL+Z) indicate that the first key should be held down while the remaining keys are pressed; all keys can then be released together.
- Words and commands typed in by the user are shown **`like this`**.
- Labels and titles on the screen are shown **like this**.
- Drop-down menu options are indicated by the name of the options separated by a vertical bar, for example **File|Print**. In this example you need to select the option **Print** from the **File** menu or tab. To do this, click when the mouse pointer is on the **File** menu or tab name; move the pointer to **Print**; when **Print** is highlighted, click the mouse button again.
- A button to be clicked will look like this.
- The names of software packages are identified *like this*, and the names of files to be used **like this**.

## Software Used

*XAMPP*

## Files Used

sqlcourse2db.sql

## Revision Information

| Version | Date | Author | Changes made |
|---|---|---|---|
| 1.0 | Oct 2014 | Mohammad Yaqub | Creation of the text |
| 1.1 | Mar 2015 | Mohammad Yaqub | Revising the whole book |
| 1.2 | Jun 2015 | Mohammad Yaqub | Changes to the exercises and removing the use of the external website |
| 1.3 | Nov 2015 | Mohammad Yaqub | Minor changes to the slides |
| 1.4 | May 2016 | Mohammad Yaqub | Minor revision to some exercises |
| 2.0 | Nov 2016 | Mohammad Yaqub | Minor changes to make the book better suited for researchers |
| 2.1 | Feb 2017 | Mohammad Yaqub | New ITLC style |
| 2.2 | May 2018 | Mohammad Yaqub | Changes to slides |
| 2.3 | Feb 2019 | Mohammad Yaqub | Minor review |
| 2.4 | Jun 2019 | Mohammad Yaqub | Changes of the path of XAMPP because of new installation setup in the lab |
| 2.5 | Oct 2019 | Mohammad Yaqub | Changed Lynda.com to LinkedIn Learning |
| 2.6 | May 2020 | Mohammad Yaqub | Changes to make the course suitable for online teaching |

## Copyright

## Acknowledgement

Some of the syntax in this book was adopted from http://dev.mysql.com.

# Contents

# Exercises

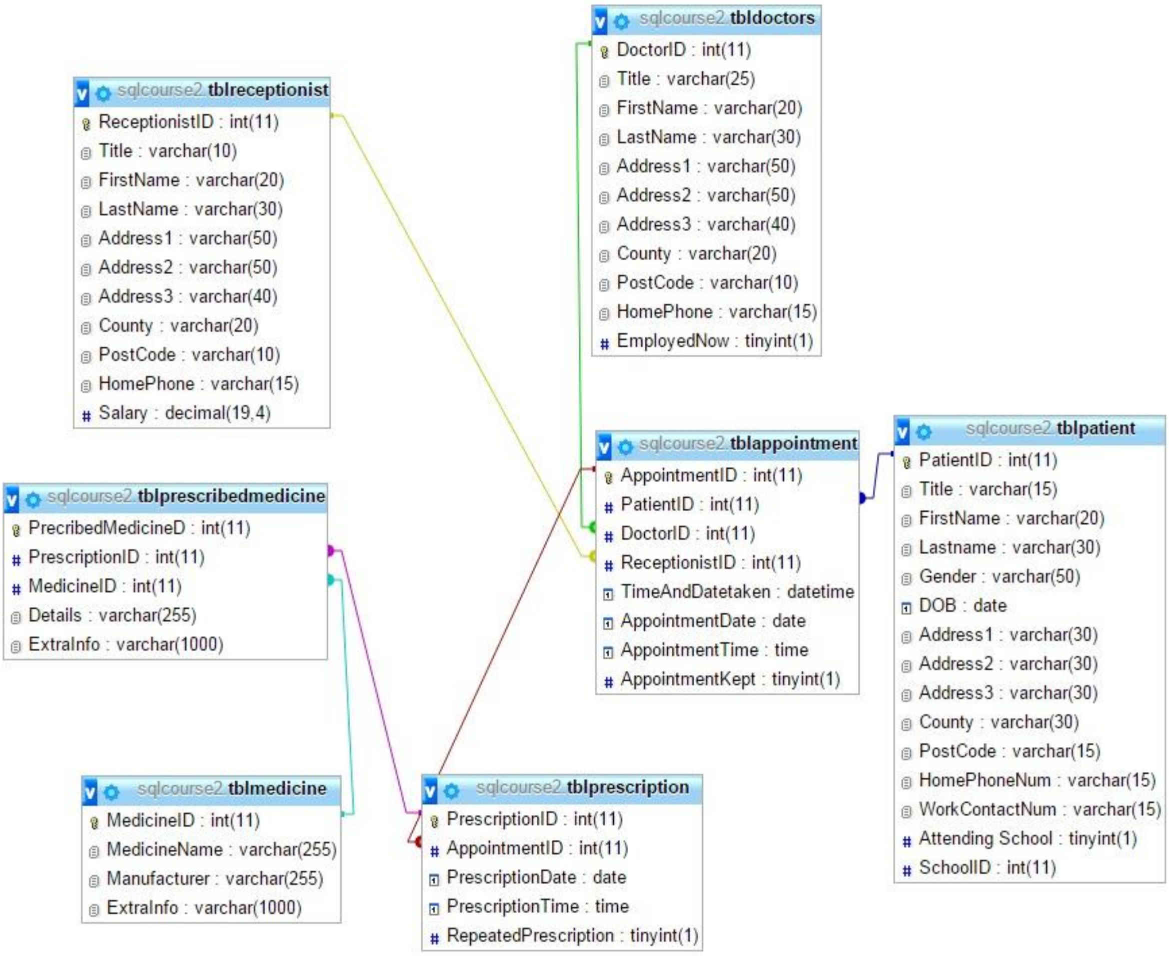
# 1 Introduction

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. How to Complete the Exercises

After installing XAMPP, you are good to go. You need to open XAMPP control panel (either open from the start menu or usually exists in C:/xampp/) and start Apache and MySQL services. 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 Centre. The database is for a surgery called St. Giles. This database contains 7 tables to hold patients, doctors, receptionists, medicines, prescriptions and appointments data. Figure 1 shows a schematic diagram of the database. The figure also shows table names (tblPatient, tblDoctors, tblReceptionist, tblMedicine, tblPrescription, tblPrescribedMedicine and tblAppointment) and field names (or columns) in each table. It also shows the data type for each field. The links between tables in the figure reflect the primary-foreign key relationships. Make sure you understand the database structure and the relationships involved.



sqlcourse2.tbldoctors
DoctorID : int(11)
Title : varchar(25)
FirstName : varchar(20)
LastName : varchar(30)
Address1 : varchar(50)
Address2 : varchar(50)
Address3 : varchar(40)
County : varchar(20)
PostCode : varchar(10)
HomePhone : varchar(15)
EmployedNow : tinyint(1)

sqlcourse2.tblreceptionist
ReceptionistID : int(11)
Title : varchar(10)
FirstName : varchar(20)
LastName : varchar(30)
Address1 : varchar(50)
Address2 : varchar(50)
Address3 : varchar(40)
County : varchar(20)
PostCode : varchar(10)
HomePhone : varchar(15)
Salary : decimal(19,4)

sqlcourse2.tblappointment
AppointmentID : int(11)
PatientID : int(11)
DoctorID : int(11)
ReceptionistID : int(11)
TimeAndDatetaken : datetime
AppointmentDate : date
AppointmentTime : time
AppointmentKept : tinyint(1)

sqlcourse2.tblpatient
PatientID : int(11)
Title : varchar(15)
FirstName : varchar(20)
Lastname : varchar(30)
Gender : varchar(50)
DOB : date
Address1 : varchar(30)
Address2 : varchar(30)
Address3 : varchar(30)
County : varchar(30)
PostCode : varchar(15)
HomePhoneNum : varchar(15)
WorkContactNum : varchar(15)
Attending School : tinyint(1)
SchoolID : int(11)

sqlcourse2.tblprescribedmedicine
PrecribedMedicineD : int(11)
PrescriptionID : int(11)
MedicineID : int(11)
Details : varchar(255)
ExtraInfo : varchar(1000)

sqlcourse2.tblmedicine
MedicineID : int(11)
MedicineName : varchar(255)
Manufacturer : varchar(255)
ExtraInfo : varchar(1000)

sqlcourse2.tblprescription
PrescriptionID : int(11)
AppointmentID : int(11)
PrescriptionDate : date
PrescriptionTime : time
RepeatedPrescription : tinyint(1)

Figure 1. St. Giles Surgery database structure.

### Exercise 1 Import St Giles database

I have created a database called *sqlcourse2db*. You are going to use this database in the rest of the exercises.

Suggested time to spend on this exercise is 8 minutes

| | |
|---|---|
| **Task 1**<br>Launch XAMPP and phpMyAdmin | **Step 1**<br>Open XAMPP Control Panel from the start menu.<br>**Step 2**<br>Start Apache and MySQL services by clicking on the top two Start buttons in XAMPP control panel.<br>**Step 3**<br>Launch a browser (Firefox or Chrome are preferred).<br>**Step 4**<br>Open phpMyAdmin by typing the following on your browser address bar<br>`Localhost/phpmyadmin` |
| **Task 2**<br>Import the database | **Step 1**<br>You have been given a file called `sqlcourse2db.sql`. The file is either on the "network drive H" if this course is face to face or given to you by the teacher if it is an online course.<br>Locate the file.<br>**Step 2**<br>You can edit its contents using any text editor like Notepad. Have a quick look by editing the sql file.<br>**Step 3**<br>From phpMyAdmin, click on the Import tab.<br>**Step 4**<br>Click on the Choose File button. Note that in some browsers the button is called Browse.<br>**Step 5**<br>Locate the file `sqlcourse2db.sql` and click Open.<br>**Step 6**<br>Click the Go button at the bottom of the page to import the sql file. |
| **Task 3**<br>Access the newly imported database | **Step 1**<br>If the import is successful, a message on the page will confirm this. You can then click on the Databases tab to see all the existing MySQL databases including `sqlcourse2db` (the database you have just imported).<br>**Step 2**<br>Click on the sqlcourse2db database link to view its contents. It should show the tables as in Figure 1. |

**Note**

From Exercise 2 onward, you can write any `SELECT` SQL statement in the SQL box provided by phpMyAdmin.

## 1.2. Recap: MySQL introduction course

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.

Structured Query Language (SQL) is a relational database language which allows you to create, delete, access and manipulate databases.

**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. In this course we will be using XAMPP which includes MySQL because it is straightforward to install and use. 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. For more details refer to the Introduction of MySQL course.

The following is the main MySQL statements you should know about:

1. Creating/Dropping SQL users

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


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

2. Creating/Dropping Databases

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

3. Creating Tables

```
CREATE TABLE tbl_name   (col_name column_definition,...)
```

The ***column_definition*** is the description of a column in the table. The minimum format of the column definition is to specify the data type e.g., INT for integer.

4. INSERT Statement

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

5. UPDATE Statement

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

### 6. DELETE Statement

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

### 7. SELECT Statement

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

### 8. Where Clause

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

### 9. Conditions

```
WHERE column_name operator value
```

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

### 10. Sorting Data – ORDER BY Clause

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

### 11. LIKE, NOT LIKE and STRCMP

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

### 12. The BETWEEN Operator

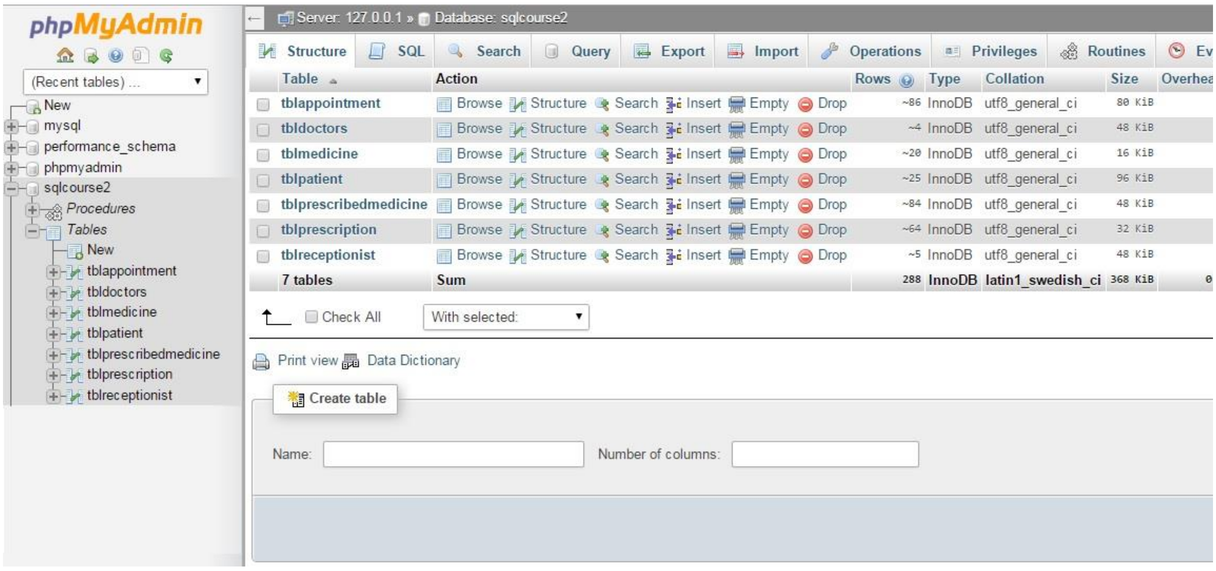
```
WHERE column_name BETWEEN value1 AND value2
```

# 2 Advanced Queries

## 2.1. Dealing with NULL Entries

Columns are filled with values. A value can be, for example, a number, a word, or a date. A special value is the NULL value. The NULL value is comparable with "value unknown" or "value not present." Use the `ISNULL()` operator to select rows that have no value in a particular column.

| **Exercise 2 Query records which have NULL values** | |
|---|---|
| Suggested time to spend on this exercise is 6 minutes | |
| **Task 1**<br>Retrieve patient records who have their work telephone number as NULL. | **Step 1**<br>Make sure you are on inside the `sqlcourse2db` database. In other words, phpMyAdmin shows that you have selected this database and you should be able to see the list of tables. If not then click on this database from the list of databases shown (as in Figure 2).<br>**Step 2**<br>Write the below `SELECT` statement in the SQL box in phpMyAdmin and see the result.<br>**Step 3**<br>Click Go. |
| `SELECT * FROM tblpatient WHERE ISNULL(WorkContactNum)` | |
| **Task 2 [Optional Task]**<br>Update patient records to change the ones that have an empty work telephone number (NULL) to the word 'NA' | **Step 1**<br>Write the following statement in the SQL box in phpMyAdmin. |
| `Update tblpatient SET WorkContactNum = 'NA' WHERE ISNULL(WorkContactNum)` | |
| | **Step 2**<br>Click on the table `tblpatient` to check if records have been updated correctly. |



| Table | Action | | | | | | Rows | Type | Collation | Size | Overhea |
|---|---|---|---|---|---|---|---|---|---|---|---|
| tblappointment | Browse | Structure | Search | Insert | Empty | Drop | ~86 | InnoDB | utf8_general_ci | 80 KiB | |
| tbldoctors | Browse | Structure | Search | Insert | Empty | Drop | ~4 | InnoDB | utf8_general_ci | 48 KiB | |
| tblmedicine | Browse | Structure | Search | Insert | Empty | Drop | ~20 | InnoDB | utf8_general_ci | 16 KiB | |
| tblpatient | Browse | Structure | Search | Insert | Empty | Drop | ~25 | InnoDB | utf8_general_ci | 96 KiB | |
| tblprescribedmedicine | Browse | Structure | Search | Insert | Empty | Drop | ~84 | InnoDB | utf8_general_ci | 48 KiB | |
| tblprescription | Browse | Structure | Search | Insert | Empty | Drop | ~64 | InnoDB | utf8_general_ci | 32 KiB | |
| tblreceptionist | Browse | Structure | Search | Insert | Empty | Drop | ~5 | InnoDB | utf8_general_ci | 48 KiB | |
| 7 tables | Sum | | | | | | 288 | InnoDB | latin1_swedish_ci | 368 KiB | 0 |

Figure 2. phpMyAdmin after importing sqlcourse2db.sql.

## 2.2. The EXISTS Operator

The EXISTS condition is used in combination with a subquery and is considered TRUE if the subquery returns at least one row. NOT EXISTS is the opposite of EXISTS and it is considered TRUE if the subquery returns no records. For example, suppose you want to retrieve all patient names that have at least one appointment within a specific date range. Then you might want to use EXISTS.

Its basic syntax is

```
WHERE EXISTS ( subquery )
```

**Exercise 3 Using the EXISTS operator**

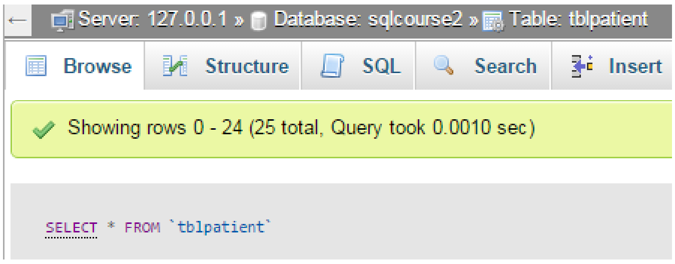
Suggested time to spend on this exercise is 4 minutes

**Task 1**

Retrieve patient first and last names who made at least one appointment

**Step 1**

Before you proceed. Check how many patient you have. You can simply click on the patient table. phpMyAdmin should show that you have 25 total.


Server: 127.0.0.1 » Database: sqlcourse2 » Table: tblpatient
Browse
Structure
SQL
Search
Insert

Showing rows 0 - 24 (25 total, Query took 0.0010 sec)

```
SELECT * FROM `tblpatient`
```

Another solution is to write a query which uses the COUNT function. See Section 2.6.

**Step 2**

Write the following statement in the phpMyAdmin SQL box.

**Step 3**

Click Go.

```
SELECT FirstName, Lastname FROM tblpatient AS pat WHERE EXISTS(select
app.PatientID FROM tblappointment AS app WHERE pat.PatientID =
app.PatientID)
```

## 2.3. 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 trying to compare a field with multiple values. For instance, you can use the equal sign to compare a column to multiple values as follows:

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

The IN operator can be used here to make the SELECT statement shorter and easier to read.

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

The second form of the IN operator can be used with a subquery. This happens when a value of a field from a one table matches one or more for another table. Try the following exercise.

**Exercise 4 Querying multiple tables: use IN operator**

Suggested time to spend on this exercise is 6 minutes

**Task 1**

Retrieve patient records that have appointments on a specific date, e.g., 2013-07-02.

**Step 1**

Write the following statement in the phpMyAdmin SQL box.

**Step 2**

Click Go.

```
SELECT * FROM      tblpatient WHERE PatientID IN (SELECT PatientID FROM
tblappointment WHERE AppointmentDate = '2013-07-02')
```

**Task 2 [Optional Task]**

Retrieve patient records who have appointments with Dr Down.

**Step 1**

Write the following statement he phpMyAdmin SQL box.

**Step 2**

Click Go.

```
SELECT * FROM      tblPatient WHERE PatientID IN (SELECT PatientID FROM
tblAppointment WHERE DoctorID IN (SELECT DoctorID FROM tblDoctors WHERE
LastName='Down'))
```

NOTE: You can use nested `IN` in a statement. However, the `IN` operator tends to be slow in execution. Therefore, `JOIN`s are a better replacement of the `IN` operator. For more information, see Section 2.4

## 2.4. JOIN operators

An SQL `JOIN` clause is used to combine rows from two or more tables, based on a common field between them. There are a few types of joins. The most common one is `INNER JOIN` or simply `JOIN`. We will only practice the `INNER JOIN` in this course but here is the definition of each one.

- `INNER JOIN` or `JOIN`: Returns all rows when there is at least one match in BOTH tables
- `LEFT JOIN`: Return all rows from the left table, and the matched rows from the right table
- `RIGHT JOIN`: Return all rows from the right table, and the matched rows from the left table
- `FULL JOIN`: Return all rows when there is a match in ONE of the tables

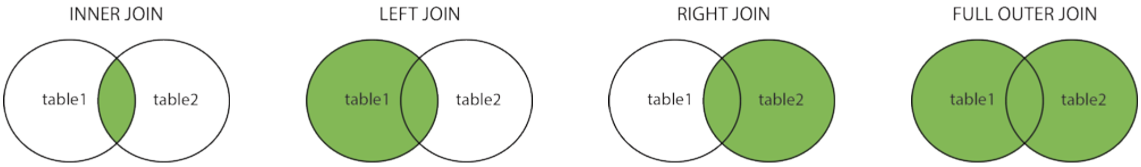


Figure 3. Graphical representation of the different types of `JOIN`s. Shaded regions represent the returned result when joining two tables.

**Exercise 5 Query multiple tables using the JOIN operator**

Suggested time to spend on this exercise is 8 minutes

| | |
|---|---|
| **Task 1**<br>Rewrite the query in Exercise 4Task 1 using the `JOIN` operator | **Step 1**<br>Write the following statement in the phpMyAdmin SQL box.<br>**Step 2**<br>Click Go. |

```
SELECT p.* FROM tblpatient AS p JOIN tblappointment AS app ON
(p.PatientID = app.PatientID AND app.AppointmentDate = '2013-07-02')
```

| | |
|---|---|
| **Task 2 [Optional Task]**<br>Rewrite the query in Exercise 4Task 2 using the `JOIN` operator. This time add doctor name and appointment date with patient details | **Step 1**<br>Write the following statement in the phpMyAdmin SQL box.<br>**Step 2**<br>Click Go. |

```
SELECT p.*, app.AppointmentDate, CONCAT(d.FirstName, ' ', d.LastName) AS
'Name of the doctor' FROM tblpatient AS p INNER JOIN tblappointment AS
app ON(p.PatientID = app.PatientID) INNER JOIN tbldoctors AS d ON
(app.DoctorID = d.DoctorID AND d.LastName='Down')
```

| | |
|---|---|
| **Task 3 [Optional Task]**<br>Write a query to retrieve the first name, last name and home phone number of patients who have been prescribed a medicine made by a company call ABC Health. | **Step 1**<br>Write the following statement in the phpMyAdmin SQL box.<br>**Step 2**<br>Click Go. |

```
SELECT p.FirstName, p.Lastname, p.HomePhoneNum FROM tblpatient AS p INNER
JOIN tblappointment AS app ON(p.PatientID = app.PatientID) INNER JOIN
tblprescription AS pre ON (app.AppointmentID = pre.AppointmentID) INNER
JOIN tblprescribedmedicine AS preM ON (pre.PrescriptionID =
preM.PrescriptionID) INNER JOIN tblMedicine AS m ON(preM.MedicineID =
m.MedicineID AND Manufacturer = 'ABC Health')
```

**Other optional tasks**

- Write a query to retrieve receptionist names who made appointments for Dr Smith
- Write a query to retrieve doctors who prescribed the medicine Tramadol
- Write a query to retrieve doctors who prescribed the medicine Tramadol from 2013-07-01 until 2013-07-05

## 2.5. REGEXP operator

`REGEXP` is a pattern matching operator in MySQL which allows simple to very detailed string comparison. It is more powerful than `LIKE` because it allows a more granular comparison. `NOT REGEXP` is the negation of `REGEXP`. Table 1 shows a list of pattern characters which we can use in comparisons.

`REGEXP` is not case sensitive, except when used with binary strings which can be done by using the keyword `BINARY` during comparison. Table 2 provides several examples of `REGEXP`.

Table 1. MySQL regular expression patterns

| **Pattern** | **What the pattern matches** |
|---|---|
| ^ | Beginning of string |
| $ | End of string |
| . | Any single character |
| [...] | Any character listed between the square brackets |
| [^...] | Any character not listed between the square brackets |
| p1\|p2\|p3 | Alternation; matches any of the patterns p1, p2, or p3 |
| * | Zero or more instances of preceding element |
| + | One or more instances of preceding element |
| {n} | n instances of preceding element |
| {m,n} | m through n instances of preceding element |

Table 2. REGEXP examples

| | |
|---|---|
| `Lastname REGEXP  '^ch'` | Matches any name which starts with ch |
| `Lastname REGEXP  'rew'` | Matches any name which contains rew |
| `Lastname REGEXP  'q$'` | Matches any name which ends with q |
| `City REGEXP '^[aeiou]w'` | Matches any city which starts with a vowel and contains the letter w |
| `City REGEXP '[^c-k][ewk]$'` | Matches any city which does NOT start with any letter from c to k and the city ends with the e, w or k |
| `phoneContactNo REGEXP '^[0-9]{10}$'` | Matches any number of 10 digits |

<table>
<tr><td colspan="2"><b>Exercise 6 Use of REGEXP</b></td></tr>
<tr><td colspan="2">Suggested time to spend on this exercise is 5 minutes</td></tr>
<tr><td><b>Task 1</b><br>Retrieve first and last names of doctors whose first names start with a vowel</td><td><b>Step 1</b><br>Write the following statement in the phpMyAdmin SQL box.<br><b>Step 2</b><br>Click Go.</td></tr>
<tr><td colspan="2">SELECT FirstName, LastName FROM tbldoctors WHERE FirstName REGEXP '^[aeiou]';</td></tr>
<tr><td colspan="2"><b>Task 2 [Optional Task]</b><br>Retrieve first and last names of doctors whose last names DO NOT start with a voxel.</td></tr>
<tr><td colspan="2"><b>Task 3 [Optional Task]</b><br>Retrieve all doctor records whose home phone is 11 digits in the form of 5 digits followed by one space followed 6 digits e.g., "01865 654896".</td></tr>
<tr><td colspan="2"><b>Task 4 [Optional Task]</b><br>Retrieve all receptionist details who live in a close not a road, e.g., address1 is <i>5 harts close</i> then you should get the record of this receptionist.</td></tr>
<tr><td colspan="2"><b>Task 5 [Optional Task]</b><br>For each prescribed medicine retrieve medicine details which were prescribed to be consumed 2 or 3 times a day only.<br>Check with the teacher for more details.</td></tr>
</table>

## 2.6. Aggregate Functions

Instead of returning a single list of data from a table, aggregate functions can be used to carry out calculations on the stored data. For instance, you can find the total number of records in a table using the `COUNT` function. Table 3 shows the common aggregate functions in MySQL. The general syntax for such functions is:

```
SELECT function(column_name) AS another_name FROM table_name
```

Table 3. List of aggregate functions in MySQL

| Function Name | Description |
|---|---|
| **AVG()** | Average value of the argument |
| **BIT_AND()** | Bitwise AND |
| **BIT_OR()** | Bitwise OR |
| **BIT_XOR()** | Bitwise XOR (exclusive or) |
| **COUNT(DISTINCT)** | Counts the number of different values |
| **COUNT()** | Counts the number of rows returned |
| **GROUP_CONCAT()** | Concatenates several strings |
| **MAX()** | The maximum value |
| **MIN()** | The minimum value |
| **STDDEV_POP()**[1] | Population standard deviation |
| **STDDEV_SAMP()**[1] | Sample standard deviation |
| **STDDEV()**[1] | Population standard deviation |
| **SUM()** | Total |
| **VAR_POP()** | Population variance |
| **VAR_SAMP()** | Sample variance |
| **VARIANCE()** | Standard variance |

[1] The difference between these three standard deviation functions is in the way the variance is computed for more information see http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/impala_stddev.html

**Exercise 7 Querying data – Aggregate Functions**

Suggested time to spend on this exercise is 7 minutes

**Task 1**

Retrieve the number of doctors in this surgery

**Step 1**

Write the following statement in the phpMyAdmin SQL box.

**Step 2**

Click Go.

```
SELECT COUNT(*) AS 'Num of doctors' FROM tblDoctors
```

**Task 2**

Write a query to find the number of female patients. Please solve this yourself. Hint: you need to use the WHERE clause in your query.

**Task 3**

Write a query to return the number of appointments between '2013-07-02' and '2013-07-04'

**Optional Tasks**

- Write a query to return the number of appointments between '2013-07-02' and '2013-07-04' for Dr Smith
- Write a query to find the receptionist who gets the highest salary.

## 2.7. Grouping records – GROUP BY Clause

The `GROUP BY` clause groups rows on the basis of similarities between them. For instance you could retrieve patients by the first letters of their post code or by gender. GROUP BY is usually used with aggregate functions. For example, to retrieve the number of male and female patients in the table `tblpatient`, we write

```
SELECT Gender, COUNT(*) from tblpatient GROUP BY Gender
```

**Exercise 8 Grouping records**

Suggested time to spend on this exercise is 8 minutes

**Task 1**

Retrieve the first and last names of all doctors and the number of appointments for each one.

**Step 1**

Write the following statement in the phpMyAdmin SQL box.

**Step 2**

Click Go.

**Step 3**

The total number of appointments is 86. Does the sum of appointments for all doctor match this?

```
SELECT d.FirstName, d.LastName, COUNT(app.DoctorID) AS 'No. of
Appointments' FROM tbldoctors AS d, tblappointment AS app WHERE d.DoctorID =
 app.DoctorID GROUP BY app.DoctorID
```

**Task 2 [Optional task]**

Retrieve the name of each medicine and the number of times it has been prescribed.

```
SELECT m.MedicineName, COUNT(p.MedicineID) FROM tblmedicine as m JOIN tblpre
scribedmedicine AS p ON (m.MedicineID = p.MedicineID) GROUP BY p.MedicineID
```

**Other Optional Tasks**

- Retrieve the name of the patient(s) who have the maximum number of appointments.
- Retrieve the name of the patient(s) who have the maximum number of appointments between '2013-07-02' and '2013-07-04'.

## 2.8. The HAVING Clause

The `WHERE` clause cannot be used in conjunction with aggregate functions. Therefore, the `HAVING` clause acts the same as the `WHERE` clause but it is only used when aggregate functions are used in conditional part of the SQL statement.

Its basic syntax is

```
SELECT * FROM TableName WHERE ColumnName operator value GROUP BY ColumnName
HAVING FunctionName(ColumnName) operator value
```

**Exercise 9 The HAVING Clause**

Suggested time to spend on this exercise is 8 minutes

| Task 1 | Step 1 |
|---|---|
| Retrieve the first and last name and number of appointments of patients who have more than three appointments booked. | Write the following statement in the phpMyAdmin SQL box.<br>**Step 2**<br>Click Go. |

```
SELECT p.FirstName, p.Lastname, COUNT(app.AppointmentID) FROM tblpatient
AS p INNER JOIN tblappointment AS app ON (p.PatientID = app.PatientID)
GROUP BY app.PatientID HAVING COUNT(app.AppointmentID) > 3
```

**Task 2 [Optional task]**

Write a query to retrieve the receptionist who made the most number of appointments.

**Task 3 [Optional task]**

Write a query to retrieve the medicine that was prescribed the most in the surgery.

**Task 4 [Optional task]**

Write a query to retrieve the doctor name who prescribed Tramadol the most between 2013-07-02 to 2013-07-08

## 2.9. Output query results to a file

So far the output of the queries we wrote either appears on the phpMyAdmin interface or in the website we created. What if we want to save the result of a query to a file? In MySQL, you can use the phrase `INTO OUTFILE` to output the result of a query to a file.

The syntax is:

```
SELECT ColumnName(s) FROM TableName INTO OUTFILE 'filename.ext'
```

Here, `ext` is the extension of the file e.g., txt or CSV.

To export the output of a query into a CVS file with column heading, you need to include extra options. The following is the general syntax for it

```
SELECT ColumnName(s)  FROM TableName INTO OUTFILE 'filename.csv' FIELDS
ENCLOSED BY '"' TERMINATED BY ';' ESCAPED BY '"' LINES TERMINATED BY
'\r\n';
```

## Exercise 10 Output data to a txt file

Suggested time to spend on this exercise is 5 minutes

**Task 1**

Output the content of the patient table to a text file called patients.txt

**Step 1**

Write the following statement in the phpMyAdmin SQL box.

**Step 2**

Click Go.

```
SELECT * FROM tblpatient INTO OUTFILE 'patients.txt'
```

**Step 3**

Check the content of the text file. Open the `xampp` folder, click on Explorer button on the XAMPP control panel. By default the file will be saved in `xampp/mysql/data/sqlcourse2db`. Open the file with a text editor like Notepad.

**Step 4**

The format of the file is not appropriate when visualised in Notepad, so copy the content on the file and paste it in an Excel sheet to get a proper table of the exported data.

**Task 2 [Optional task]**

Try to export other queries to a CSV format.

# 3 Procedures, Functions & Triggers

## 3.1. Stored Procedures

A stored procedure is a certain piece of code (the procedure) consisting of declarative and procedural SQL statements stored in the catalogue of a database that can be activated by calling it from a program, a trigger, or another stored procedure.

Its basic syntax is

```
CREATE PROCEDURE Proc_name
        ([proc_parameter[,...]])
BEGIN
Proc_body which contains valid SQL statement(s)
END
```

The complete syntax for creating stored procedures has more options which you might want to explore further. For more information, visit http://dev.mysql.com/.

The reason why we might need to write a procedure is for example we need to execute a specific set of SQL statements at different occasions. For instance, you may want to write a procedure which changes the salary of a specific receptionist. Yes, you can write one `UPDATE` statement to do this but why not to write this statement in a procedure which can be executed at different times by calling its name and sending the appropriate parameters to it, e.g., receptionist ID and the new salary.

Once the procedure is created, you can execute it using the `CALL` command. The syntax is:

```
CALL Proc_name (parameters)
```

Delimiters other than the default “;” are typically used when defining functions, stored procedures, and triggers. The delimiter can be any set of characters you choose, but it needs to be a distinctive set of characters that won't cause further confusion. The reason we need to use a delimiter when creating a function, procedure or trigger is because the whole function for example has to appear for MySQL as one statement. If we don’t change the default delimiter then any statement inside a function for example will end the function before it actually ends. Therefore, when creating a new function, procedure or trigger, we can change the delimiter to allow MySQL statements within these blocks to use the semicolon inside them without creating confusion with the real END of the block.

## Exercise 11 Creating a database stored procedure

Suggested time to spend on this exercise is 8 minutes

**Task 1**

Create a stored procedure which allows updating the salary of a receptionist given the receptionist ID.

**Step 1**

Write the following statement in the phpMyAdmin SQL box.

**Step 2**

Click Go.

```
DELIMITER $$

CREATE PROCEDURE changeSalary (IN rID INT, IN newSalary INT)

BEGIN

  Update tblreceptionist SET Salary = newSalary WHERE ReceptionistID =
rID;

END$$

DELIMITER ;
```

This procedure is called changeSalary. It receives two parameters (rID of the receptionist and the new salary of the receptionist). The procedure will execute one UPDATE statement to change the salary of the receptionist (given her/his id) to the new salary.

**Task 2**

Execute the procedure by changing the salary from 12500 to 13000 for the receptionist Mrs Avery whose ID is 1.

**Step 1**

Before you change the salary, open the receptionist table on phpMyAdmin and check Mrs Avery's salary. It should be 12500.

**Step 2**

Write the following statement in the phpMyAdmin SQL box.

**Step 3**

Click Go.

```
CALL changeSalary(1,13000)
```

**Task 3 [optional task]**

Execute the procedure on a different receptionist

**Step 1**

Choose any receptionist and change her salary. For instance, the code below changes the salary of receptionist Burns (ID is 4) from 12000 to 15000.

```
CALL changeSalary(4,15000)
```

## 3.2. Functions

A function (also called a routine) is the same as a procedure but it can return a value. We typically call the function using the SELECT statement. Its basic syntax is:

```
CREATE FUNCTION Fun_name
        ([fun_parameter[,...]])
        RETURNS type
BEGIN
        Function_body which contains valid SQL statement(s)
END
```

**Exercise 12 Creating a database function**

Suggested time to spend on this exercise is 8 minutes

| Task | Step |
|---|---|
| **Task 1**<br>Create a function which retrieves the number of appointments made by a receptionist given the receptionist ID. | **Step 1**<br>Write the following statement in the phpMyAdmin SQL box. |

```
DELIMITER $$
CREATE FUNCTION getNumApp (rID INT)
     RETURNS INT
BEGIN
    DECLARE NumofApp INT;
    SELECT COUNT(AppointmentID) INTO NumofApp FROM tblappointment WHERE
    ReceptionistID = rID;
    RETURN NumofApp;
end$$
DELIMITER ;
```

| Task | Step |
|---|---|
| **Task 2**<br>Call the function to retrieve the number of appointments made by receptionist 2. | **Step 1**<br>Write the following statement in the phpMyAdmin SQL box. |

```
SELECT getNumApp(2)
```

**Task 3 [optional task]**

Create a function which retrieves the number of appointments made by a receptionist given the receptionist last name. Also call the function to test if it is working.

## 3.3. Triggers

A trigger is a piece of code consisting of procedural and declarative statements stored in the catalogue and activated by the database server if a specific operation is executed on the database, and only then when a certain condition holds. The way triggers are called deviates from that of stored procedures. Triggers cannot be called explicitly, either from a program or from a stored procedure. There is no `CALL` or `EXECUTE TRIGGER` statement or similar statement available. Triggers are called by SQL engine itself, without the programs or users being aware of it. Calling triggers is transparent to users.

A trigger is called by SQL when a program, interactive user, or stored procedure executes a specific database operation, such as adding a new row to a table or removing all rows. So, triggers are executed automatically by SQL, and it is impossible to activate triggers from a program. It is also impossible to "switch off" triggers from a program unless it is completely removed.

The basic syntax of creating a trigger is:

```
CREATE TRIGGER trigger_name
    { BEFORE | AFTER } { INSERT | UPDATE | DELETE }
    ON tbl_name FOR EACH ROW
BEGIN
    trigger_body
END
```

## Exercise 13 Creating a trigger

Suggested time to spend on this exercise is 10 minutes

**Task 1**

Create an empty backup table for the patient table

**Step 1**

Write the following statement in the phpMyAdmin SQL box.

```
CREATE TABLE tblpatientbackup LIKE tblpatient
```

**Task 2**

Create a trigger which checks if a patient record is deleted from tblpatient then before deleting the record, it gets backed up on the patient back up table `tblpatientbackup`

**Step 1**

Write the following statement in the phpMyAdmin SQL box.

```
DELIMITER $$

CREATE TRIGGER backupPatient BEFORE DELETE ON tblpatient FOR EACH ROW

BEGIN

INSERT   INTO   tblpatientbackup   VALUES   (OLD.PatientID,   OLD.Title,
OLD.FirstName,   OLD.Lastname,   OLD.Gender,   OLD.DOB,   OLD.Address1,
OLD.Address2,  OLD.Address3,OLD.County,  OLD.PostCode,  OLD.HomePhoneNum,
OLD.WorkContactNum, OLD.`Attending School` , OLD.SchoolID);

END$$

DELIMITER ;
```

**Task 3**

Now, let's delete a record or more from tblpatient

**Step 1**

Before you actually delete anything, open the table `tblpatient` to check if there is a patient with ID =4

**Step 2**

Also, open `tblpatientbackup` and check if it is empty.

**Step 3**

Now let us delete a record. Write the following statement in the phpMyAdmin SQL box.

```
DELETE FROM tblpatient WHERE PatientID = 4
```

**Task 4**

Check if the record was deleted from `tblpatient` and it is copied to `tblpatientback` table.

**Task 5 [optional task]**

Try to delete multiple records from `tblpatient` using one `DELETE` statement then check if all records are copied over to `tblpatientbackup`

**Task 6 [optional task]**

Change the trigger to add the date and time of deletion

**Task 7 [optional task]**

Change the trigger to add the date, time and the user how deleted the records

# 4 Advanced concepts: quick visit

## 4.1. Indexes

SQL has several methods of accessing rows in a table. The two best known are the sequential access method (also called scanning or browsing) and the indexed access method.

The sequential access method browses through a table row by row. Each row in a table is read. If only one row has to be found in a table with many rows, this method is, of course, very time-consuming and inefficient.

When SQL uses the indexed access method, it reads only the rows that exhibit the required characteristics. To do this, however, an index is necessary. An index is a type of alternative access to a table and can be compared with the index in a book. Creating an index for a specific column in a table makes searching this field fast. However, this will create a space overhead which requires creating an index table for this column. Creating an index for each column in each table is not appropriate since the size of the database will increase significantly. For more information search for MySQL index.

## 4.2. Cursors

To handle a result set inside a stored procedure, you use a cursor. A cursor allows you to iterate on a set of rows returned by a query and process each row accordingly.

## 4.3. Storage Engines

Storage engines are MySQL components that handle the SQL operations for different table types. MySQL supports several storage engines that have differences in handling data in tables. The current default engine is InnoDB. However there are a few other storage engines like MyISAM (used to be the default), NDB, Memory, Example, CVS, etc. For more information, visit http://dev.mysql.com/doc/refman/5.5/en/storage-engines.html. Table 4 shows a few major differences between MyISAM and InnoDB.

Table 4. MyISAM vs InnoDB

| MyISAM | InnoDB |
|---|---|
| Not Atomicity, Consistency, Isolation, Durability (ACID) compliant and non-transactional | ACID compliant and hence fully transactional with ROLLBACK and COMMIT and support for Foreign Keys |
| MySQL Default Engine for versions before 5.5 | MySQL Default Engine for versions 5.5 and newer |
| Offers Compression | Offers Compression |
| Requires full repair/rebuild of indexes/tables after crash | Auto recovery from crash via logs checking |
| No ordering in storage of data | Row data stored in pages in primary key order |
| Table level locking | Row level locking |

## 4.4. INFORMATION_SCHEMA

INFORMATION_SCHEMA provides access to database metadata which contains information about the database such as the name of a database or table, the data type of a column, or access privileges. For instance, the following statement retrieve the list of all databases in MySQL:

```
show databases
```

To show the list of tables in a database:

```
show tables
```

To show all tables in all databases in MySQL:

```
SELECT table_name, table_schema FROM information_schema.tables
```

To list all columns in all tables in MySQL:

```
SELECT column_name  FROM information_schema.columns
```

# 5 What is Next?

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

## 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!**

# Data Management: Databases - MySQL Further Techniques for Researchers

Mohammad Yaqub

mohammad.yaqub@it.ox.ac.uk

# Ready To Learn?

- Today’s session takes place in a video-call using *Teams*
- Can you see and hear the teacher?
- Please tell us if anything doesn’t work
- Don’t plan to multi-task

# Today's arrangements

You will have: Class notes

Schedule

|  |  |
|---|---|
| 09:30 – 10:10 | Teaching |
| 10:30 – 11:30 | Practice session |
| 13:00 – 13:30 | Teaching |
| 14:00 – 14:50 | Practice session |

# Today’s resources

- How will you display your workbook?
- Where are your course files?
- Is the software installed?

🔇 📵

# Do at home - MySQL Installation

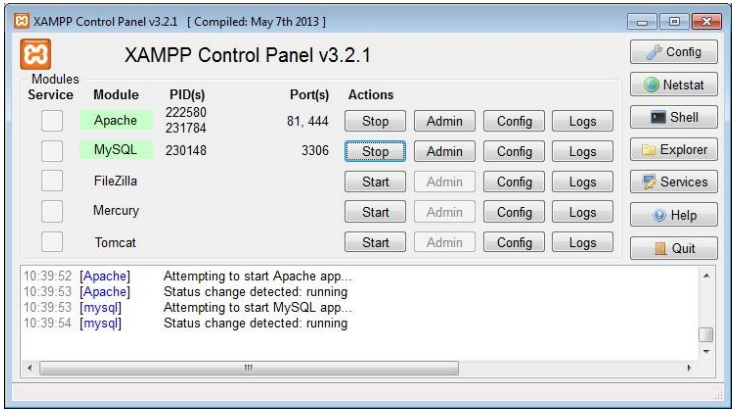
- MySQL is free
- It can be installed from different sources
- Recommended sources
  - XAMPP, WAMP, MAMP, AMPPS, etc : PHP + MySQL
  - MySQL workbench
- In XAMPP, phpMyAdmin is used as an interface to control MySQL. Command line can be used as well.
- phpMyAdmin: a web-based *frontend* interface for MySQL.

# Do at home - MySQL Installation

- Download XAMPP https://www.apachefriends.org/download.html

  Choose the first version of XAMPP according to the operating system you are using

- Install XAMPP: usually installed in
  - Windows: `c:/xampp`
  - Mac: Applications
- Open XAMPP control panel (from XAMPP top directory or the start menu in Windows)
- From XAMPP control panel, start Apache and MySQL services



XAMPP Control Panel v3.2.1 [ Compiled: May 7th 2013 ]

XAMPP Control Panel v3.2.1

Modules

| Service | Module | PID(s) | Port(s) | Actions | | | |
|---|---|---|---|---|---|---|---|
| □ | Apache | 222580<br>231784 | 81, 444 | Stop | Admin | Config | Logs |
| □ | MySQL | 230148 | 3306 | Stop | Admin | Config | Logs |
| □ | FileZilla | | | Start | Admin | Config | Logs |
| □ | Mercury | | | Start | Admin | Config | Logs |
| □ | Tomcat | | | Start | Admin | Config | Logs |

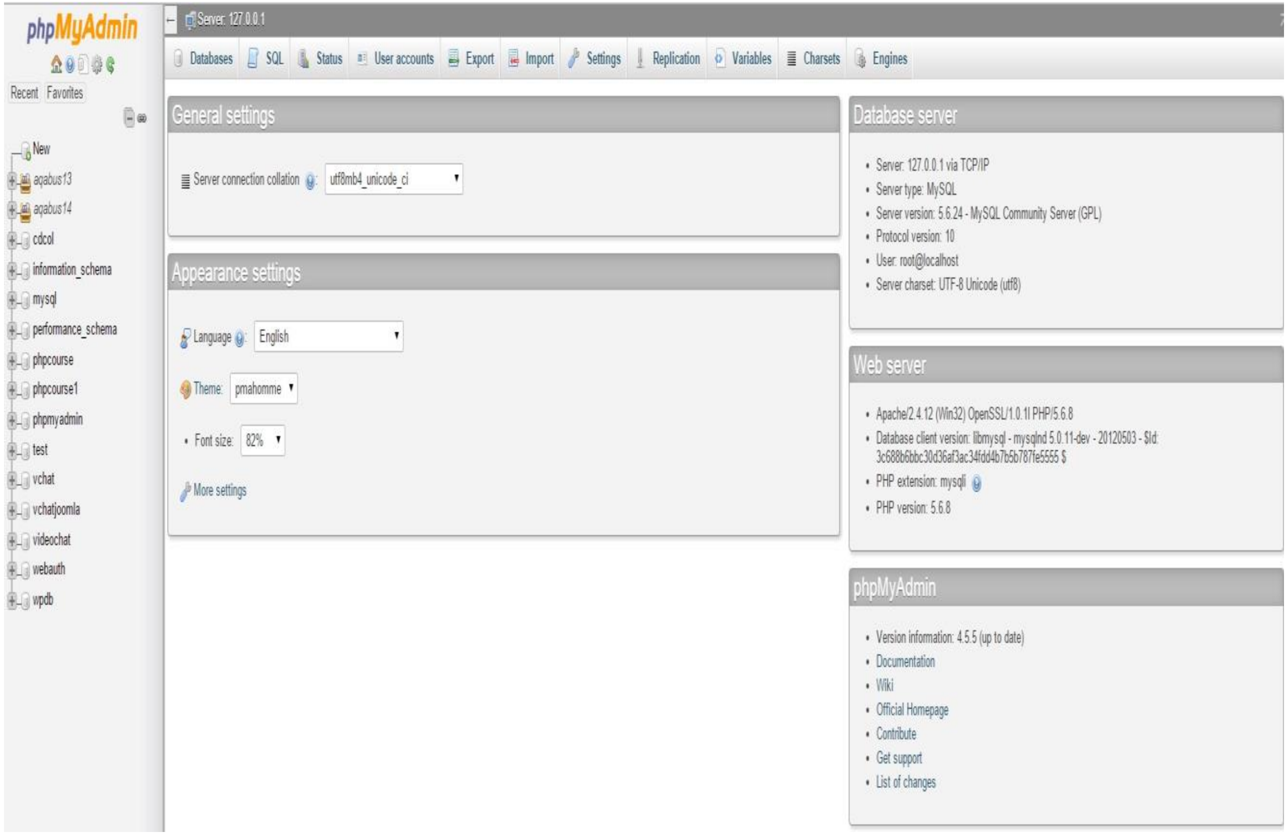
```
10:39:52  [Apache]    Attempting to start Apache app...
10:39:53  [Apache]    Status change detected: running
10:39:53  [mysql]     Attempting to start MySQL app...
10:39:54  [mysql]     Status change detected: running
```

# Do at home - MySQL Installation and testing

- Open any browser (preferably Chrome or Firefox) and type the following as is (don’t select if Chrome makes a suggestion for instance)

```
localhost/phpmyadmin
```

- You should see a page like this. If you do, you are good to go



General settings

Server connection collation: utf8mb4_unicode_ci

Appearance settings

Language: English

Theme: pmahomme

- Font size: 82%

More settings

Database server

- Server: 127.0.0.1 via TCP/IP
- Server type: MySQL
- Server version: 5.6.24 - MySQL Community Server (GPL)
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8)

Web server

- Apache/2.4.12 (Win32) OpenSSL/1.0.1l PHP/5.6.8
- Database client version: libmysql - mysqlnd 5.0.11-dev - 20120503 - $Id: 3c688b6bbc30d36af3ac34fdd4b7b5b787fe5555 $
- PHP extension: mysqli
- PHP version: 5.6.8

phpMyAdmin

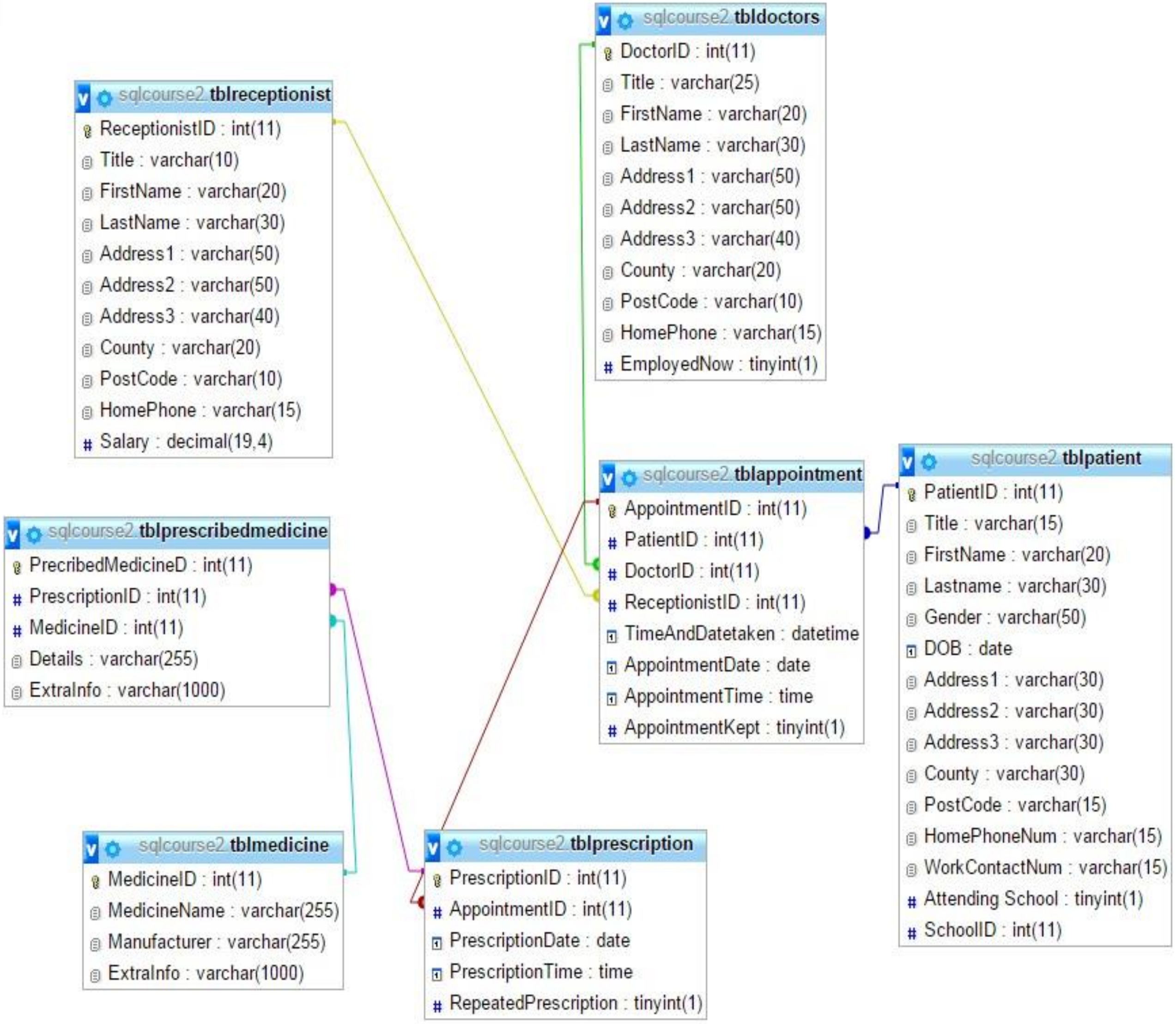
- Version information: 4.5.5 (up to date)
- Documentation
- Wiki
- Official Homepage
- Contribute
- Get support
- List of changes

# Introduction to MySQL: revisit

- The following concepts are prerequisites to this course:
- Creating/Dropping SQL users
- Creating Tables
- INSERT, UPDATE & DELETE Statements
- SELECT Statement
- Where Clause, Conditions & ORDER BY Clause
- String comparisons: LIKE, NOT LIKE and STRCMP
- The BETWEEN Operator
- XAMPP & phpMyAdmin

# How to complete the exercises

- Start Apache and MySQL from XAMPP control panel
- To access phpMyAdmin, open a browser and type `localhost/phpmyadmin`



| sqlcourse2.tblreceptionist |
|---|
| ReceptionistID : int(11) |
| Title : varchar(10) |
| FirstName : varchar(20) |
| LastName : varchar(30) |
| Address1 : varchar(50) |
| Address2 : varchar(50) |
| Address3 : varchar(40) |
| County : varchar(20) |
| PostCode : varchar(10) |
| HomePhone : varchar(15) |
| Salary : decimal(19,4) |

| sqlcourse2.tbldoctors |
|---|
| DoctorID : int(11) |
| Title : varchar(25) |
| FirstName : varchar(20) |
| LastName : varchar(30) |
| Address1 : varchar(50) |
| Address2 : varchar(50) |
| Address3 : varchar(40) |
| County : varchar(20) |
| PostCode : varchar(10) |
| HomePhone : varchar(15) |
| EmployedNow : tinyint(1) |

| sqlcourse2.tblprescribedmedicine |
|---|
| PrecribedMedicineD : int(11) |
| PrescriptionID : int(11) |
| MedicineID : int(11) |
| Details : varchar(255) |
| ExtraInfo : varchar(1000) |

| sqlcourse2.tblappointment |
|---|
| AppointmentID : int(11) |
| PatientID : int(11) |
| DoctorID : int(11) |
| ReceptionistID : int(11) |
| TimeAndDatetaken : datetime |
| AppointmentDate : date |
| AppointmentTime : time |
| AppointmentKept : tinyint(1) |

| sqlcourse2.tblpatient |
|---|
| PatientID : int(11) |
| Title : varchar(15) |
| FirstName : varchar(20) |
| Lastname : varchar(30) |
| Gender : varchar(50) |
| DOB : date |
| Address1 : varchar(30) |
| Address2 : varchar(30) |
| Address3 : varchar(30) |
| County : varchar(30) |
| PostCode : varchar(15) |
| HomePhoneNum : varchar(15) |
| WorkContactNum : varchar(15) |
| Attending School : tinyint(1) |
| SchoolID : int(11) |

| sqlcourse2.tblmedicine |
|---|
| MedicineID : int(11) |
| MedicineName : varchar(255) |
| Manufacturer : varchar(255) |
| ExtraInfo : varchar(1000) |

| sqlcourse2.tblprescription |
|---|
| PrescriptionID : int(11) |
| AppointmentID : int(11) |
| PrescriptionDate : date |
| PrescriptionTime : time |
| RepeatedPrescription : tinyint(1) |

Check Figure 1 (Page 7)

# Topics

- Dealing with NULL values
- EXISTS operator
- IN operator
- JOINs
- Regular expression
- Aggregate functions
- HAVING clause
- GROUP BY clause
- Output query results to a file
- Database procedures, functions and triggers

# Advanced Queries – Dealing with NULL

- Some cells in a table can be empty or NULL.
- NULL is comparable with "value unknown" or "value not present."
- In MySQL, `ISNULL()` can be used to check if a cell is NULL, for example:

```
SELECT * FROM tblpatient WHERE
ISNULL(Address2)
```

# Advanced Queries –EXISTS Operator

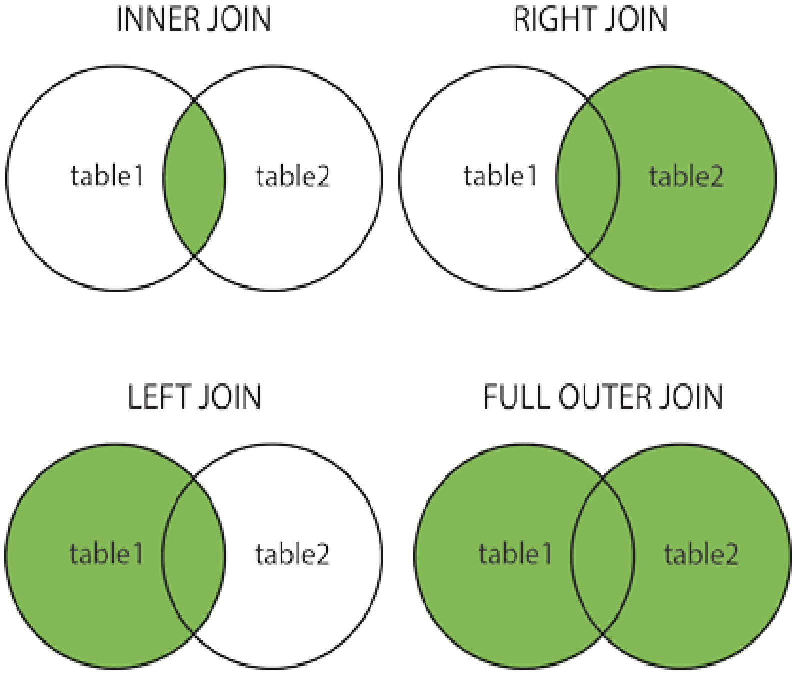
- `EXISTS` is used in combination with a subquery and it is considered TRUE if the subquery returns at least one row.
- Its syntax: … `WHERE EXISTS ( subquery )`
- `Not EXISTS`
- Example: Select all receptionist who made at least one appointment

```
SELECT r.FirstName, r.LastName FROM
tblreceptionist AS r WHERE EXISTS(SELECT
app.AppointmentID FROM tblappointment AS app
WHERE r.ReceptionistID = app.ReceptionistID)
```

# Advanced Queries – JOINs

- JOINs are used to combine rows from two or more tables, based on a common field between them.
- JOINs types:
  - `INNER JOIN` or `JOIN`
  - `LEFT JOIN`
  - `RIGHT JOIN`
  - `FULL OUTER JOIN`

    (not supported in MySQL but can be emulated using `Left` and `Right` Joins and a `UNION` operator)



# Inner Join

- Examples:
- Write a query to retrieve the patients who have appointments from 2013-07-01 to 07-05

```
SELECT p.FirstName, p.LastName, app.AppointmentDate FROM
tblpatient AS p JOIN tblappointment AS app ON (p.PatientID =
app.PatientID AND app.AppointmentDate BETWEEN '2013-07-01' AND
'2013-07-05')
```

- What does the following query retrieve?

```
SELECT d.FirstName, d.LastName, app.AppointmentDate, p.LastName
FROM tbldoctors AS d JOIN tblappointment AS app ON (d.DoctorID
= app.DoctorID) JOIN tblpatient AS p ON (app.PatientID =
p.PatientID) ORDER BY p.PatientID
```

# String comparison using REGEXP

- `REGEXP` is a pattern matching operator in MySQL which allows simple to very detailed string comparison.

| MySQL regular expression patterns | |
|---|---|
| **Pattern** | What the pattern matches |
| **^** | Beginning of string |
| **$** | End of string |
| **.** | Any single character |
| **[...]** | Any character listed between the square brackets |
| **[^...]** | Any character not listed between the square brackets |
| **p1\|p2\|p3** | Alternation; matches any of the patterns p1, p2, or p3 |
| ***** | Zero or more instances of preceding element |
| **+** | One or more instances of preceding element |
| **{n}** | n instances of preceding element |
| **{m,n}** | m through n instances of preceding element |

# String comparison using REGEXP

- You can also use NOT REGEXP
- Examples

| REGEXP examples | |
|---|---|
| **Lastname REGEXP  '^ch'** | Matches any name which starts with “ch” |
| **Lastname REGEXP  'rew'** | Matches any name which contains “rew” |
| **Lastname REGEXP  'q$'** | Matches any name which ends with “q” |
| **City REGEXP '^[aeiou]w'** | Matches any city which starts with a vowel and the second letter is “w” |
| **City REGEXP ‘^[^c-k][ewk]$'** | Matches any city which does NOT start with any letter from c to k and the city ends with “e”, “w” or “k” |
| **phoneContactNo REGEXP '^[0-9]{10}$'** | Matches any number of 10 digits |

# Aggregate Functions

- A few functions can be applied to a table data

**List of the common aggregate functions in MySQL**

| Function Name | Description |
|---|---|
| AVG() | Return the average value of the argument |
| COUNT(DISTINCT) | Return the count of a number of different values |
| COUNT() | Return a count of the number of rows returned |
| MAX() | Return the maximum value |
| MIN() | Return the minimum value |
| STDDEV() | Return the population standard deviation |
| SUM() | Return the sum |
| VARIANCE() | Return the population standard variance |

- Examples

```
SELECT COUNT(*) FROM tblPatient
```

```
SELECT SUM(Salary) FROM tblReceptionist WHERE FirstName
REGEXP '^[LPS]'
```

# Practice time

Exercises 1 – 7

Spend 60 minutes

# GROUP BY Clause

- `GROUP BY` is usually used with aggregate functions.
- It groups rows on the basis of similarities between them.
- Example: retrieve the number of male and female patients

```
SELECT COUNT(*) from tblpatient

SELECT COUNT(*) from tblpatient GROUP BY Gender

SELECT Gender, COUNT(*) from tblpatient GROUP BY Gender
```

# The HAVING Clause

- MySQL does not allow the use of aggregate functions in the `WHERE` clause.
- The `HAVING Clause` works as an alternative of `WHERE` in case an aggregate function is used in the condition.
- Example: Get receptionist details who made more than 15 appointments

```
SELECT r.*, COUNT(app.AppointmentID) FROM
tblreceptionist AS r JOIN tblappointment AS app ON
(r.ReceptionistID = app.ReceptionistID) GROUP BY
app.ReceptionistID HAVING COUNT(app.AppointmentID)>15
```

# Output query results to a file

- `INTO OUTFILE` can be used in a MySQL query to output its result to a file.
- Example: Output patient details to a file called pat.txt

```
SELECT * FROM tblpatient INTO OUTFILE 'pat.txt'
```

- By default, XAMPP saves the file to `c:/xampp/mysql/data/db_name` unless you make an absolute path.
- However, in today’s XAMPP installation, to get the XAMPP folder, click the **Explorer** button on the XAMPP control panel
- You need write access to the folder you are saving to.

# Stored Procedures

- A stored procedure is a certain piece of code
- It consists of declarative and procedural SQL statements
- It is stored in the catalogue of a database
- It can be activated by calling it from a program, a trigger, or another stored procedure.
- It basic syntax

```
CREATE PROCEDURE Proc_name ([proc_parameter[,...]])
BEGIN
Proc_body which contains valid SQL statement(s)
END;
```

# Creating a stored Procedure - Example

```
DELIMITER $$

CREATE PROCEDURE  GetDocApp ( IN  docID INT, IN  d
DATE, OUT numApp INT )

BEGIN
   SELECT COUNT( AppointmentID )
   INTO numApp
   FROM tblappointment
   WHERE DoctorID = docID
   AND AppointmentDate = d;

END$$

DELIMITER ;
```

- To call (execute) this procedure

```
CALL GetDocApp(1,'2013-07-03', @n);

SELECT @n;
```

# Creating Functions

- A function (also called a routine) is the same as a procedure but it can return a value.
- It basic syntax

```
CREATE FUNCTION Fun_name ([fun_parameter[,...]])
  RETURNS type
BEGIN
  Function_body which contains valid SQL statement(s)
END;
```

# Creating a function - Example

```
DELIMITER $$
CREATE FUNCTION GetDocAppFun (docID INT, d DATE)
RETURNS INT
BEGIN
 DECLARE numApp INT;
 SELECT COUNT( AppointmentID ) INTO numApp FROM
 tblappointment WHERE DoctorID = docID AND
 AppointmentDate = d;
 RETURN numApp;
END $$
DELIMITER ;
```

- To call this function

```
SELECT GetDocAppFun(1,'2013-07-03');
```

# Triggers

- A trigger is a piece of code consisting stored in the database
- It gets executed only if a specific operation is executed on the database
- It cannot be called explicitly like a stored procedure
- Creating a trigger basic syntax

```
CREATE TRIGGER trigger_name
    { BEFORE | AFTER } { INSERT | UPDATE | DELETE }
    ON tbl_name FOR EACH ROW
BEGIN
    trigger_body
END
```

# Creating a trigger- Example

```
DELIMITER $$

CREATE TRIGGER backupPatient BEFORE DELETE ON
tblpatient FOR EACH ROW

BEGIN

  INSERT INTO tblpatientbackup VALUES (OLD.PatientID,
  OLD.Title, OLD.FirstName, OLD.Lastname, OLD.Gender,
  OLD.DOB, OLD.Address1, OLD.Address2,
  OLD.Address3,OLD.County, OLD.PostCode,
  OLD.HomePhoneNum, OLD.WorkContactNum, OLD.`Attending
  School` , OLD.SchoolID);

END$$

DELIMITER ;
```

Note: in the code above, ` is used to wrap the name of a field which is made of two words

# Practice time

Exercises 8 – 13

The teacher will be available 50 minutes

If you have any questions in future, please email the teacher

# Advanced Topics

- Indexes
- Cursors
- Storage Engines
- INFORMATION_SCHEMA
- DBA tasks
- Backups & Logs
- `EXPLAIN` statement
- Third party tools e.g., Workbench, phpMyAdmin, Maatkit, Sequel Pro, Navicat, etc.

# Where to find some help

- Google is your friend
- http://dev.mysql.com
- www.w3schools.com/sql/
- www.stackoverflow.com

- **LinkedIn Learning**

# LinkedIn Learning

- Formerly known as Lynda.com
- It is free
- It is in support of the IT Learning Centre activities
- ITLC will still offer a full range of classroom-based courses
- ITLC is happy to work with people around the University to help them use LinkedIn Learning effectively in their departments and colleges
- https://help.it.ox.ac.uk/courses/molly

# Other courses

- IT Learning centre course:

https://skills.it.ox.ac.uk/courses-home

- MySQL Introduction
- Programming
  - Concepts
  - Perl
  - Python
  - C++
  - Java
  - **JavaScript**
  - MATLAB
  - **PHP**

- IT Learning Portfolio https://skills.it.ox.ac.uk/it-learning-portfolio

# Please respond to the feedback survey

mohammad.yaqub@it.ox.ac.uk

iT Learning Centre

iT services

UNIVERSITY OF OXFORD