Chapter 7: MYSQL More to GO

Objectives:

After reading this chapter you should be entertained and learned to:

1- Using operators in MYSQL.
2- Hierarchy of Conditions in Select Statement.
3- Working with Dates.
4- String Functions.
5- Knowing Some Numeric Functions.
6- Using PHPMyAdmin
7- Having Handy MYSQL Commands.

7.1 Introduction:

The previous chapter concerned with the fundamental concepts. Although it may help you to develop a complete system, you need to use the concepts that would be discussed in this chapter. This chapter will help you to develop more practical applications with more facilities.

7.2 More about Operators:

Operators are another tool that you can use within your MYSQL statement to refine your search for specific records.

7.2.1 Logical Operators:

```
SELECT * FROM spareparts WHERE ((tire = ‘105’) AND (spark = '305'));
```

The above statement uses the '**AND**' operator (it can also be expressed as '**&&**') to combine two conditions. Both conditions have to be met in order for the record to be displayed. We can also use the 'OR' operator (can be expressed as '||') to ask for a record to be displayed if either condition is met.

```
SELECT * FROM spareparts WHERE ((tire = ‘105’) or (spark = '405'));
```

The final operator we'll discuss here is the '**NOT**' operator ('**!**' in case you were wondering), which is a bit more complicated. Rather than joining conditions together it becomes part of the condition, turning a positive into a negative. The following statement retrieves all records that do *not* contain '**oil**' as '**208**'.

```
SELECT * FROM spareparts WHERE (oil != ‘108’);
```

As the 'NOT' operator has become part of the condition it can be used with another operator to combine positive and negative conditions.

7.2.2 Numeric Operators:

Numeric operators are used to compare two numbers under several conditions (shown in Table 7.1).

Table 7.1 Numerical Operators.

Operator	Condition
=	Equal to
<>	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Let us look at the “spareparts” table in “order300”:

```
mysql> select * from spareparts;
```

This will show all rows into the table “spareparts”.

```
+------+------+-------+
| tire | oil  | spark |
+------+------+-------+
|  101 |  201 |   301 |
|  102 |  202 |   302 |
|  103 |  203 |   303 |
|  104 |  204 |   304 |
|  105 |  205 |   305 |
|  106 |  206 |   306 |
|  107 |  207 |   307 |
|  108 |  208 |   308 |
|  109 |  209 |   309 |
|  110 |  210 |   310 |
|  111 |  222 |   333 |
|    0 |    0 |     0 |
|    0 |    0 |     0 |
|    0 |    0 |     0 |
+------+------+-------+
14 rows in set (0.33 sec)
```

```
mysql> Delete from spareparts where tire < 100;
```

This would be equivalent to:

```
mysql> Delete from spareparts where tire = 0;
```

```
Query OK, 3 rows affected (0.06 sec)
```

So, that command deleted the zero records.

7.2.3 Other Operators:

Let us prepare the car table in chapter 6.

```
mysql> describe cars;
```

```
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| CarID        | int(3)      | YES  |     | NULL    |       |
| Manufacturer | varchar(10) | YES  |     | NULL    |       |
| Year         | int(4)      | YES  |     | NULL    |       |
| Car          | varchar(10) | YES  |     | NULL    |       |
| CC           | int(4)      | YES  |     | NULL    |       |
| AirCon       | int(1)      | YES  |     | NULL    |       |
| CDMulti      | int(1)      | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
7 rows in set (0.05 sec)
```

```
mysql> load data infile 'cars.txt'
          replace into table cars
          fields terminated by '\t'
          lines terminated by '\n'
          (CarID,Manufacturer,Year,Car,CC,AirCon,CDMulti);
```

This output:

```
Query OK, 4 rows affected (0.11 sec)
Records: 4  Deleted: 0  Skipped: 0  Warnings: 0
mysql> select * from cars;
```

This will show all rows into the table “cars”.

```
+-------+--------------+------+---------+------+--------+---------+
| CarID | Manufacturer | Year | Car     | CC   | AirCon | CDMulti |
+-------+--------------+------+---------+------+--------+---------+
|  1205 | Nasr         | 2005 | 128     | 1300 |      0 |       0 |
|  1206 | Nasr         | 2006 | Sahin   | 1600 |      0 |       0 |
|  1207 | Toyota       | 2007 | Corolla | 1300 |      1 |       1 |
|  1208 | Nissan       | 2007 | Sunny   | 1300 |      1 |       1 |
+-------+--------------+------+---------+------+--------+---------+
4 rows in set (0.00 sec)
```

There are two other operators so far unmentioned, the LIKE and BETWEEN operators.

7.2.3.1 LIKE Operator:

The LIKE operator is used when we want to match part of the data in a field by using the '**%**' wildcard. So, if we wanted to search the 'Manufacturer' field for all ones beginning with '**N**', we would use the following statement.

```
mysql> SELECT * FROM cars WHERE (Manufacturer LIKE 'N%');
```

This will show all rows into the table “cars” which manufacturer starts with “N”.

```
+-------+--------------+------+-------+------+--------+---------+
| CarID | Manufacturer | Year | Car   | CC   | AirCon | CDMulti |
+-------+--------------+------+-------+------+--------+---------+
|  1205 | Nasr         | 2005 | 128   | 1300 |      0 |       0 |
|  1206 | Nasr         | 2006 | Sahin | 1600 |      0 |       0 |
|  1208 | Nissan       | 2007 | Sunny | 1300 |      1 |       1 |
+-------+--------------+------+-------+------+--------+---------+
3 rows in set (0.13 sec)
```

The 'N' in the search string is followed by our wildcard character. To match the word 'the' in any 'Manufacturer' we would put a wildcard character before and afterwards in the SQL statement.

7.2.3.2 BETWEEN Operator:

The BETWEEN statement is used to add more functionality to a condition, allowing us to select a range of values from a column.

```
mysql> SELECT * FROM cars WHERE (year BETWEEN 2005 AND 2006);
```

This will show all rows into the table “cars” which year value lies between 2005 and 2006.

```
+-------+--------------+------+-------+------+--------+---------+
| CarID | Manufacturer | Year | Car   | CC   | AirCon | CDMulti |
+-------+--------------+------+-------+------+--------+---------+
|  1205 | Nasr         | 2005 | 128   | 1300 |      0 |       0 |
|  1206 | Nasr         | 2006 | Sahin | 1600 |      0 |       0 |
+-------+--------------+------+-------+------+--------+---------+
2 rows in set (1.48 sec)
```

Or we could select an alphanumeric range. To select all “Manufacturer” alphabetically between “Toyota” and “Nissan”.

7.2.4 Data Types:

Creating a table requires you to have an understanding of MYSQL data types (similar if not identical to other database data types) so that you can define the fields in the table. The data types below (shown in table 7.2) are some of the basic data types and are meant as a simple introduction. If you are planning a production database you should investigate the data types thoroughly.

Table 7.2 Data Types in MYSQL.

Data type	**Description**	**Example:**
INT	Numeric entry	id INT
VARCHAR(n)	Text string of characters up to **n** with a maximum of 255 characters	name VARCHAR(20)
CHAR(n)	Text string with specific number (n) of characters. If the number of characters is less than '**n**' then is padded by spaces (spaces are removed when data is retrieved). 255 Maximum.	address CHAR(30)
TEXT	Holds between 255 - 65535 characters	philosophy TEXT
DATE	The date stored in the format YYYY-MM-DD	dob DATE
TIME	The time stored in the format HH:MM:SS	tob TIME

The syntax for defining a field and data type (as used in the examples) is 'fieldname data type'.

7.3 Hierarchy of Conditions in Select Statement:

The conditions that control the filtering of the query exist in a rough (if not quite exact) hierarchal order:

```
              WHERE
                   |
             GROUP BY
             |       |
   HAVING       ORDER BY
             |       |
             LIMIT
```

If more than one of these conditions are to used in a query then it is recommended that this order be used, so that GROUP BY cannot go before the WHERE condition and LIMIT cannot go before HAVING etc. Notice that HAVING and ORDER BY are at the same level - this is because either one can go before the other. Breaking this hierarchy will generally result in errors. We have already looked at the WHERE condition extensively, so let us have a closer look at these conditions.

7.3.1 GROUP BY:

When we issue a SELECT command we are shown the results in the order that the records were entered. We can change this by using the 'GROUP BY' directive, allowing us to display the data grouped by field. To return the results ordered by artist:

```
mysql> SELECT * FROM cars GROUP BY Year;
```

This will show all rows into the table “cars” grouped by “Year”.

```
+-------+--------------+------+---------+------+--------+---------+
| CarID | Manufacturer | Year | Car     | CC   | AirCon | CDMulti |
+-------+--------------+------+---------+------+--------+---------+
|  1205 | Nasr         | 2005 | 128     | 1300 |      0 |       0 |
|  1206 | Nasr         | 2006 | Sahin   | 1600 |      0 |       0 |
|  1207 | Toyota       | 2007 | Corolla | 1300 |      1 |       1 |
+-------+--------------+------+---------+------+--------+---------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM cars GROUP BY AirCon;
```

This will show all rows into the table “cars” grouped by “AirCon”.

```
+-------+--------------+------+---------+------+--------+---------+
| CarID | Manufacturer | Year | Car     | CC   | AirCon | CDMulti |
+-------+--------------+------+---------+------+--------+---------+
|  1205 | Nasr         | 2005 | 128     | 1300 |      0 |       0 |
|  1207 | Toyota       | 2007 | Corolla | 1300 |      1 |       1 |
+-------+--------------+------+---------+------+--------+---------+
2 rows in set (0.02 sec)
```

7.3.2 ORDER BY:

Similar to GROUP BY, this condition further allows control over the result set. ORDER BY controls the sequence of results. For example to display all the data from the 'car' table ordered alphabetically by 'Manufacturer' we would issue this command.

```
mysql> SELECT * FROM cars ORDER BY Manufacturer;
```

This will show all rows into the table "cars" by 'Manufacturer' in alphabetical ascending order.

```
+-------+--------------+------+---------+------+--------+---------+
| CarID | Manufacturer | Year | Car     | CC   | AirCon | CDMulti |
+-------+--------------+------+---------+------+--------+---------+
|  1205 | Nasr         | 2005 | 128     | 1300 |      0 |       0 |
|  1206 | Nasr         | 2006 | Sahin   | 1600 |      0 |       0 |
|  1208 | Nissan       | 2007 | Sunny   | 1300 |      1 |       1 |
|  1207 | Toyota       | 2007 | Corolla | 1300 |      1 |       1 |
+-------+--------------+------+---------+------+--------+---------+
4 rows in set (0.00 sec)
```

You can also choose to sort the results in reverse by appending DESC to the condition

```
mysql> SELECT * FROM cars ORDER BY Manufacturer DESC;
```

This will show all rows into the table "cars" by 'Manufacturer' in alphabetical descending order.

```
+-------+--------------+------+---------+------+--------+---------+
| CarID | Manufacturer | Year | Car     | CC   | AirCon | CDMulti |
+-------+--------------+------+---------+------+--------+---------+
|  1207 | Toyota       | 2007 | Corolla | 1300 |      1 |       1 |
|  1208 | Nissan       | 2007 | Sunny   | 1300 |      1 |       1 |
|  1205 | Nasr         | 2005 | 128     | 1300 |      0 |       0 |
|  1206 | Nasr         | 2006 | Sahin   | 1600 |      0 |       0 |
+-------+--------------+------+---------+------+--------+---------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM cars ORDER BY Car DESC;
```

```
+-------+--------------+------+---------+------+--------+---------+
| CarID | Manufacturer | Year | Car     | CC   | AirCon | CDMulti |
+-------+--------------+------+---------+------+--------+---------+
|  1208 | Nissan       | 2007 | Sunny   | 1300 |      1 |       1 |
|  1206 | Nasr         | 2006 | Sahin   | 1600 |      0 |       0 |
|  1207 | Toyota       | 2007 | Corolla | 1300 |      1 |       1 |
|  1205 | Nasr         | 2005 | 128     | 1300 |      0 |       0 |
+-------+--------------+------+---------+------+--------+---------+
4 rows in set (0.00 sec)
```

7.3.3 HAVING:

The HAVING condition is really just another WHERE condition that acts as a 'secondary constraint' on the result set. This works best when you are trying to apply a restrictive condition after a grouping has taken place. So for example:

```
mysql> SELECT * FROM cars GROUP BY Car HAVING Manufacturer LIKE
'N%';
```

This will show all rows into the table “cars” by ‘Car’ having ‘Manufacturer’ starts with ‘N’ in alphabetical ascending order.

```
+-------+--------------+------+-------+------+--------+---------+
| CarID | Manufacturer | Year | Car   | CC   | AirCon | CDMulti |
+-------+--------------+------+-------+------+--------+---------+
|  1205 | Nasr         | 2005 | 128   | 1300 |      0 |       0 |
|  1206 | Nasr         | 2006 | Sahin | 1600 |      0 |       0 |
|  1208 | Nissan       | 2007 | Sunny | 1300 |      1 |       1 |
+-------+--------------+------+-------+------+--------+---------+
3 rows in set (0.00 sec)
```

You should try if at possible to write queries using the WHERE condition rather than the HAVING condition as HAVING is un-optimized.

7.3.4 LIMIT:

There will be times that you create a query that produces a large result set that you don't want to view or handle all at once. In these situations it is useful to use the LIMIT condition to restrict the number or records returned. For example to display the first 5 records that meet a query:

```
mysql> SELECT * FROM cars LIMIT 2;
```

This will show the first two rows from the table “cars”.

```
+-------+--------------+------+-------+------+--------+---------+
| CarID | Manufacturer | Year | Car   | CC   | AirCon | CDMulti |
+-------+--------------+------+-------+------+--------+---------+
|  1205 | Nasr         | 2005 | 128   | 1300 |      0 |       0 |
|  1206 | Nasr         | 2006 | Sahin | 1600 |      0 |       0 |
+-------+--------------+------+-------+------+--------+---------+
2 rows in set (0.24 sec)
```

To retrieve the rest of the records in batches we can also specify a starting point before the number or rows to return.

```
SELECT <fields> FROM <TABLE> LIMIT <starting_point>, <number_of_rows>
```

So to return rows 2-3 we would specify the start point of 2 and then ask to return 2 records.

```
mysql> SELECT * FROM cars LIMIT 1, 2;
```

This will show the two rows from the table “cars” starting after row 1.

```
+-------+--------------+------+---------+------+--------+---------+
| CarID | Manufacturer | Year | Car     | CC   | AirCon | CDMulti |
+-------+--------------+------+---------+------+--------+---------+
|  1206 | Nasr         | 2006 | Sahin   | 1600 |      0 |       0 |
|  1207 | Toyota       | 2007 | Corolla | 1300 |      1 |       1 |
+-------+--------------+------+---------+------+--------+---------+
2 rows in set (0.00 sec)
```

7.4 Working with Dates:

Change the column year to be a date.

```
mysql> alter table cars change year prodate date;
```

This will produce:

```
Query OK, 4 rows affected (0.13 sec)
Records: 4  Duplicates: 0  Warnings: 4)
```

```
mysql> select * from cars;
```

This will show all records, but the new field is date displayed 0000-00-00.

```
+-------+--------------+------------+---------+------+--------+---------+
| CarID | Manufacturer | prodate    | Car     | CC   | AirCon | CDMulti |
+-------+--------------+------------+---------+------+--------+---------+
|  1205 | Nasr         | 0000-00-00 | 128     | 1300 |      0 |       0 |
|  1206 | Nasr         | 0000-00-00 | Sahin   | 1600 |      0 |       0 |
|  1207 | Toyota       | 0000-00-00 | Corolla | 1300 |      1 |       1 |
|  1208 | Nissan       | 0000-00-00 | Sunny   | 1300 |      1 |       1 |
+-------+--------------+------------+---------+------+--------+---------+
4 rows in set (0.00 sec)
```

So, it is preferred to update that field to have real dates.

```
mysql> select * from cars;
```

This will show all records, after making updates.

```
+-------+--------------+------------+---------+------+--------+---------+
| CarID | Manufacturer | prodate    | Car     | CC   | AirCon | CDMulti |
+-------+--------------+------------+---------+------+--------+---------+
|  1205 | Nasr         | 2005-09-16 | 128     | 1300 |      0 |       0 |
|  1206 | Nasr         | 2006-02-16 | Sahin   | 1600 |      0 |       0 |
|  1207 | Toyota       | 2007-01-16 | Corolla | 1300 |      1 |       1 |
|  1208 | Nissan       | 2008-03-19 | Sunny   | 1300 |      1 |       1 |
+-------+--------------+------------+---------+------+--------+---------+
4 rows in set (0.00 sec)
```

A typical entry for a date might look something like this.

```
mysql> select prodate from cars;
```

This will show all records, after making updates.

```
+------------+
| prodate    |
+------------+
| 2005-09-16 |
| 2006-02-16 |
| 2007-01-16 |
| 2008-03-19 |
+------------+
4 rows in set (0.00 sec)
```

You could imagine trying to SELECT dates using '**Less Than**' or '**More Than**' would be difficult as a date is not a sequential decimal number. However that is why we specify the 'DATE' data type when creating the field in the table, as this allows comparisons of this type. What actually happens is that the MySQL DBMS converts any date into the number of days since year '0' before doing any comparison.

Try SELECTing prodate that was produced prior to the year 2006.

```
mysql> select * from cars where (prodate < '2006-00-00');
```

This will show one record.

```
+-------+--------------+------------+-------+------+--------+---------+
| CarID | Manufacturer | prodate    | Car   | CC   | AirCon | CDMulti |
+-------+--------------+------------+-------+------+--------+---------+
|  1205 | Nasr         | 2005-09-16 | 128   | 1300 |      0 |       0 |
+-------+--------------+------------+-------+------+--------+---------+
1 row in set (0.00 sec)
```

```
mysql> select * from cars where (prodate >= '2006-00-00');
```

This will show three records.

```
+-------+--------------+------------+---------+------+--------+---------+
| CarID | Manufacturer | prodate    | Car     | CC   | AirCon | CDMulti |
+-------+--------------+------------+---------+------+--------+---------+
|  1206 | Nasr         | 2006-02-16 | Sahin   | 1600 |      0 |       0 |
|  1207 | Toyota       | 2007-01-16 | Corolla | 1300 |      1 |       1 |
|  1208 | Nissan       | 2008-03-19 | Sunny   | 1300 |      1 |       1 |
+-------+--------------+------------+---------+------+--------+---------+
3 rows in set (0.00 sec)
```

Thus far when retrieving stored data, we have simply displayed the results of any query. MySQL can do more that this and has many built in functions that can transform data to meet our requirements. These include:

- **Date Functions** - used to manipulate the display format of a date as well as calculate time.
- **String Functions** - can manipulate a text string
- **Numeric Functions** - can manipulate figures
- **Summarising Functions** - output meta results from a query

There are also **Control Functions** that can be used to give conditionality to queries.

7.4.1 Date Functions:

Before looking at the date functions in detail it is worth revisiting the various date datatypes to gain a better understanding of the limitations of date formatting.

7.4.2 Date Datatypes:

There are 5 MySQL date datatypes (shown in table 7.3):

Table 7.3 Date dataTypes

Data type	**Format**	**Info**
DATETIME	YYYY-MM-DD HH:MM:SS	This stores both date and time.
DATE	YYYY-MM-DD	This only stores the date
TIMESTAMP(length)	Varies	See Below
TIME	HH:MM:SS	This stores only the time
YEAR	YYYY	Stores only the year

The timestamp datatype is somewhat different as it stores the time that a row was last changed. The format also varies according to the length. For example to store the same information as DATETIME, you would specify a length of 14 whereas to store the DATE you would specify a length of 8 (shown in table 7.4).

Table 7.4 TIMESTAMP format

Timestamp Definition	**Format**
TIMESTAMP(2)	YY
TIMESTAMP(4)	YYYY
TIMESTAMP(6)	YYMMDD
TIMESTAMP(8)	YYYYMMDD
TIMESTAMP(10)	YYMMDDHHMM
TIMESTAMP(12)	YYMMDDHHMMSS
TIMESTAMP(14)	YYYYMMDDHHMMSS

In the **'cars'** table we have used the DATE for the ‘prodate' field.

```
mysql> select Manufacturer, prodate from cars;
```

This will result.

```
+--------------+------------+
| Manufacturer | prodate    |
+--------------+------------+
| Nasr         | 2005-09-16 |
| Nasr         | 2006-02-16 |
| Toyota       | 2007-01-16 |
| Nissan       | 2008-03-19 |
+--------------+------------+
4 rows in set (0.00 sec)
```

So to begin with let's look at how we can manipulate these dates using MySQL's date functions.

7.4.3 DATE_FORMAT():

This function allows the developer to format the date anyway that they wish by specifying a sequence of format strings. A string is composed of the percentage symbol '**%**' followed by a letter that signifies how you wish to display part of the date. Table 7.5 shows some of the more common strings to use:

Table 7.5 The more common strings that being used by Date data type

String	**Displays**	**Example**
%d	The numeric day of the month	01....10....17....24 etc
%D	The day of the month with a suffix	1st, 2nd, 3rd.... etc
%m	The numeric month	01....04....08....11 etc
%M	The Month name	January....April....August etc
%b	The Abbreviated Month Name	Jan....Apr....Aug....Nov etc
%y	Two digit year	98, 99, 00, 01, 02, 03 etc
%Y	Four digit year	1998, 2000, 2002, 2003 etc
%W	Weekday name	Monday.... Wednesday....Friday etc
%a	Abbreviated Weekday name	Mon....Wed....Fri etc
%H	Hour (24 hour clock)	07....11....16....23 etc
%h	Hour (12 hour clock)	07....11....04....11 etc
%p	AM or PM	AM....PM
%i	Minutes	01....16....36....49 etc
%s	Seconds	01....16....36....49 etc

There are more, but that should be enough for now. There are a couple of things to note. Upper and Lowercase letters in the string make a difference and also that when arranging these strings into a sequence you can intersperse 'normal' characters. For example:

The sequence '**%d/%m/%y**', with forward slashes separating the strings, would be displayed as **01/06/03**.

The next stage is to use the function DATE_FORMAT() to convert a stored time to a format we want.

7.4.3.1 Syntax:

```
DATE_FORMAT(date, sequence)
```

Thus to change the format of the **cds.bought** field to DD-MM-YYYY we specify the field as the date and the sequence as **'%d-%m-%Y'.**

```
DATE_FORMAT(prodate, '%d-%m-%Y')
```

This function is then incorporated into our SQL statement in place of the exiting cds.bought field.

```
mysql> SELECT Manufacturer, DATE_FORMAT(prodate, '%d-%m-%Y')
FROM cars;
```

This will result.

```
+--------------+---------------------------------+
| Manufacturer | DATE_FORMAT(prodate, '%d-%m-%Y') |
+--------------+---------------------------------+
| Nasr         | 16-09-2005                      |
| Nasr         | 16-02-2006                      |
| Toyota       | 16-01-2007                      |
| Nissan       | 19-03-2008                      |
+--------------+---------------------------------+
4 rows in set (0.33 sec)
```

Dates can also be formatted in 'plain English'.

```
mysql> SELECT Manufacturer, DATE_FORMAT(prodate, '%W the %D of
     %M %Y') FROM cars;
```

This will result.

```
+--------------+-------------------------------------------+
| Manufacturer | DATE_FORMAT(prodate, '%W the %D of %M %Y') |
+--------------+-------------------------------------------+
| Nasr         | Friday the 16th of September 2005         |
| Nasr         | Thursday the 16th of February 2006        |
| Toyota       | Tuesday the 16th of January 2007          |
| Nissan       | Wednesday the 19th of March 2008          |
+--------------+-------------------------------------------+
4 rows in set (0.00 sec)
```

Note: DATE_FORMAT() only works with data types that include the date. This means DATE, DATETIME and TIMESTAMP. There is a similar function called TIME_FORMAT() that works with TIME as well as DATETIME and TIMESTAMP.

7.4.3.2 Extraction Functions:

As well as using DATE_FORMAT() there are other functions that allow you to extract specific information about a date (year, month, day etc) – Shown in table 7.6. These include:

Table 7.6 Extraction Functions

Function	**Displays**	**Example**
DAYOFMONTH(date)	The numeric day of the month	01....10....17....24 etc
DAYNAME(date)	The Name of the day	Monday.... Wednesday....Friday etc
MONTH(date)	The numeric month	01....04....08....11 etc
MONTHNAME(date)	The Month name	January....April....August etc
YEAR(date)	Four digit year	1998, 2000, 2002, 2003 etc
HOUR(time)	Hour (24 hour clock)	07....11....16....23 etc

MINUTE(time)	Minutes	01....16....36....49 etc
SECOND(time)	Seconds	01....16....36....49 etc
DAYOFYEAR(date)	Numeric day of the year	1.....366

To give an example of one of these you can use DAYNAME() to work out which day you were born on. To do this you can specify the date directly to the function without referring to any tables or field. So for my birthday (20th July 1973):

```
mysql> SELECT DAYNAME('1958-09-16');
```

This will result.

```
+-----------------------+
| DAYNAME('1958-09-16') |
+-----------------------+
| Tuesday               |
+-----------------------+
1 row in set (0.00 sec)
```

Or you could even SELECT two or three date items.

```
mysql>  SELECT  DAYNAME('1973-10-06'),MONTHNAME('1973-10-06'),
        YEAR('1973-10-06');
```

This will result.

```
+-----------------------+-------------------------+--------------------+
| DAYNAME('1973-10-06') | MONTHNAME('1973-10-06') | YEAR('1973-10-06') |
+-----------------------+-------------------------+--------------------+
| Saturday              | October                 |               1973 |
+-----------------------+-------------------------+--------------------+
1 row in set (0.00 sec)
```

7.4.4 Getting the Current Date and Time:

There are three functions that you can use to get the current date and time. NOW() - which gets both date and time, CURDATE() which works with only the date and CURTIME() for the time.

```
mysql> SELECT NOW(), CURTIME(), CURDATE();
```

This will result.

```
+---------------------+-----------+------------+
| NOW()               | CURTIME() | CURDATE()  |
+---------------------+-----------+------------+
| 2006-09-12 22:16:25 | 22:16:25  | 2006-09-12 |
+---------------------+-----------+------------+
1 row in set (0.06 sec)
```

7.4.5 Changing Date Values:

There are two functions that allow you to add and subtract time to a date. These are DATE_ADD() and DATE_SUB().

7.4.5.1 Syntax:

```
DATE_SUB(date,INTERVAL expr type)
```

The **date** - is a standard DATE or DATETIME value, next come the command **INTERVAL** followed by the time period (**expr**) and finally what type period it is (Month, Day, Year etc). Therefore to work out the date 60 days in the future:

```
mysql> SELECT DATE_ADD(CURDATE(), INTERVAL 60 DAY);
```

This will result.

```
+--------------------------------------+
| DATE_ADD(CURDATE(), INTERVAL 60 DAY) |
+--------------------------------------+
| 2006-11-11                           |
+--------------------------------------+
1 row in set (0.00 sec)
```

Or 6 months in the past:

```
mysql> SELECT DATE_SUB(CURDATE(), INTERVAL 6 MONTH);
```

This will result.

```
+---------------------------------------+
| DATE_SUB(CURDATE(), INTERVAL 6 MONTH) |
+---------------------------------------+
| 2006-03-12                            |
+---------------------------------------+
1 row in set (0.01 sec)
```

We can also format this result as well using DATE_FORMAT().

```
mysql> SELECT DATE_FORMAT(DATE_SUB(CURDATE(), INTERVAL 6 MONTH),
        '%W the %D of % M %Y') AS 'Six Months Ago';
```

This will result.

```
+---------------------------------------+
| DATE_SUB(CURDATE(), INTERVAL 6 MONTH) |
+---------------------------------------+
| 2006-03-12                            |
+---------------------------------------+
1 row in set (0.01 sec)
```

By now you should have the idea and thus I'm not going to carry on and extensively cover all the functions, the MYSQL manual is probably the best place to look to see all the date functions.

7.5 String Functions:

String values are can be explained as 'bits of text' and much like the date functions, the string functions allow us to manipulate these values before they are displayed. Although there are once more many different functions, I'm going to concentrate on the functions that fall into a few broad categories.

- Adding text to an existing value
- Changing Part of a String
- Extracting Text from a String
- Finding a piece of text in a string

7.5.1 Adding Text to An Existing Value:

There are two simple ways to add more text to an existing value - either at the start or end of the text. Placing the text at either end is best achieved with the CONCAT() function.

7.5.1.1 Syntax:

```
CONCAT(string1,string2,...)
```

Thus we can take an existing value (say **string2**) and place a new value (**string1**) at the beginning to get **string1string2**.

```
mysql> SELECT Manufacturer from cars where CarId='1207';
```

This will result.

```
+--------------+
| Manufacturer |
+--------------+
| Toyota       |
+--------------+
1 row in set (0.00 sec)
```

If we wanted to add the text "**is the first car in Japan**." at the end.

```
mysql> SELECT CONCAT(Manufacturer," is the first car in Japan")
       FROM cars WHERE CarId='1207';
```

This will result.

```
+--------------------------------------------------+
| CONCAT(Manufacturer," is the first car in Japan") |
+--------------------------------------------------+
| Toyota is the first car in Japan                 |
+--------------------------------------------------+
1 row in set (0.09 sec)
```

Or if we wanted to say "**The best Japanese car is** " at the beginning.

```
mysql> SELECT CONCAT("The best Japanese car is ", Manufacturer)
       FROM cars WHERE CarId='1207';
```

This will result.

```
+--------------------------------------------------+
| CONCAT("The best Japanese car is ", Manufacturer) |
+--------------------------------------------------+
| The best Japanese car is Toyota                  |
+--------------------------------------------------+
1 row in set (0.00 sec)
```

7.5.2 Changing Part of a String:

As well as add text we can replace it or overwrite it completely. To replace an instance of text within a string we can use the REPLACE() function.

7.5.2.1 Syntax:

```
REPLACE(whole_string,to_be_replaced,replacement)
```

Therefore if we wanted to replace the CC '**1300**' with the CC '1299' in the **cars**:

```
mysql> SELECT CarID, Manufacturer, REPLACE(CC,'00','01') FROM
       cars WHERE CC='1300';
```

This will result. (no change in the database table contents, the change is in the displayed data).

```
+-------+--------------+-----------------------+
| CarID | Manufacturer | REPLACE(CC,'00','01') |
+-------+--------------+-----------------------+
|  1206 | Nasr         | 1601                  |
+-------+--------------+-----------------------+
1 row in set (0.00 sec)
```

7.5.3 Inserting Part of a String:

Another Function we can use to add text is the INSERT() function that overwrites any text in the string from a start point for a certain length.

7.5.3.1 Syntax:

```
INSERT(string,start_position,length,newstring)
```

In this case the crucial bits of information are the position to start (how many characters from the begriming) and the length. So again to replace '**Nasr**' (which starts at character 1 in the string) with '**EgyNast**' in the Manufacturer we need to start at position **1** for a length of **7.**

```
mysql> SELECT INSERT(Manufacturer,1,3,'Egy') FROM cars WHERE
       CarID='1206';
```

This will result.

```
+--------------------------------+
| INSERT(Manufacturer,1,3,'Egy') |
+--------------------------------+
| Egyr                           |
+--------------------------------+
1 row in set (0.00 sec)
```

N.B.: no change in the database table contents, the change is in the displayed data.

If we alter the position (say to **3**) you can see that the exchange will work in a different way.

```
mysql> SELECT INSERT(Manufacturer,5,3,'Egy') FROM cars WHERE
       CarID='1206';
```

This will result.

```
+--------------------------------+
| INSERT(Manufacturer,5,3,'Egy') |
+--------------------------------+
| NasrEgy                        |
+--------------------------------+
1 row in set (0.00 sec)
```

7.5.4 Extracting Text from a String:

As well as adding text to a string we can also use functions to extract specific data from a string. To begin with lets look at three **LEFT()**, **RIGHT()** and **MID()**.

7.5.4.1 Syntax:

```
LEFT(string,length)
RIGHT(string,length)
MID(string,start_position,length)
```

The first two, **LEFT()** and **RIGHT()**, are fairly straight forward. You specify the string and the length of the string to keep, relative to either the left or right depending on which function you are using. So to keep the words '**The**' (which occupies **3** characters on the left) and '**Album**' (**5** characters on the right) we would specify:

```
mysql> SELECT LEFT(Manufacturer,2), IGHT(Manufacturer,2),
       MID(Manufacturer,3,2) FROM cars WHERE CarID='1207';
```

This will result.

```
+----------------------+-----------------------+-----------------------+
| LEFT(Manufacturer,2) | RIGHT(Manufacturer,2) | MID(Manufacturer,3,2) |
+----------------------+-----------------------+-----------------------+
| To                   | ta                    | yo                    |
+----------------------+-----------------------+-----------------------+
1 row in set (0.00 sec)
```

7.5.5 Substring Text from a String:

There is also another extraction function that is worth mentioning; **SUBSTRING()**.

7.5.5.1 Syntax:

```
SUBSTRING(string,position)
```

This returns all of the string after the position. Thus to return ‘ota' you would start at '**4**'.

```
mysql> SELECT SUBSTRING(Manufacturer,4) FROM cars WHERE
       CarID='1207';
```

This will result.

```
+---------------------------+
| SUBSTRING(Manufacturer,4) |
+---------------------------+
| ota                       |
+---------------------------+
1 row in set (0.01 sec)
```

This returns all of the string after the position. Thus to return ‘Toy' you would start at '1' with length=3.

```
mysql> SELECT SUBSTRING(Manufacturer,1,3) FROM cars WHERE
       CarID='1207';
```

This will result.

```
+-----------------------------+
| SUBSTRING(Manufacturer,1,3) |
+-----------------------------+
| Toy                         |
+-----------------------------+
1 row in set (0.00 sec)
```

7.5.6 Finding a piece of text in a string:

In some of the string functions we have seen so far it has been necessary to provide a starting position as part of the function This position can be found using the LOCATE() function specifying the text to find (**substring**) as well as the **string** to search in.

7.5.6.1 Syntax:

```
LOCATE(substring,string)
```

So to find the location of '**ota**':

```
mysql> SELECT LOCATE('ota', Manufacturer) FROM cars WHERE
       CarID='1207';
```

This will result.

```
+-----------------------------+
| LOCATE('ota', Manufacturer) |
+-----------------------------+
|                           4 |
+-----------------------------+
1 row in set (0.00 sec)
```

If a substring is not present then '0' is returned.

```
mysql> SELECT LOCATE('ata', Manufacturer) FROM cars WHERE
       CarID='1207';
```

This will result.

```
+-----------------------------+
| LOCATE('ata', Manufacturer) |
+-----------------------------+
|                           0 |
+-----------------------------+
1 row in set (0.00 sec)
```

7.5.7 Finding the length of a string:

It is also possible to automatically calculate the length of a piece of text using **LENGTH()**.

7.5.7.1 Syntax:

```
LENGTH(string)
```

So with the word '**yota**'.

```
mysql> SELECT LENGTH('mysql> SELECT LOCATE(ata, Manufacturer)
       FROM cars WHERE CarID=1207;');
```

This will result.

```
+-------------------------------------------------------------------------------+
| LENGTH('mysql> SELECT LOCATE(ata, Manufacturer) FROM cars WHERE CarID=1207;') |
+-------------------------------------------------------------------------------+
|                                                                            67 |
+-------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

7.5.8 Combining of Some Functions:

Therefore with these results who these can be combined with one of the other functions. For example with the **MID()** function.

```
mysql> SELECT ID(car,LOCATE('yo',Manufacturer),LENGTH('Toyota'))
       FROM cars WHERE CarId='1207';
```

This will result.

```
+---------------------------------------------------+
| MID(car,LOCATE('yo',Manufacturer),LENGTH('Toyota')) |
+---------------------------------------------------+
| rolla                                             |
+---------------------------------------------------+
1 row in set (0.00 sec)
```

The LENGTH() of '**Toyota**' is worked out, the position of '**Toyota**' is worked out using LOCATE() and these values are included within the MID() function. The result is that Toyota is returned.

7.5.9 Transforming Strings:

The first two change the case of the string to either uppercase - **UCASE()** - or to lowercase - **LCASE()**

.

7.9.1 Syntax:

```
LCASE(string)
UCASE(string)
```

As you can imagine the usage of these are fairly straightforward.

```
mysql> SELECT LCASE(Manufacturer), UCASE(Manufacturer) FROM cars
       WHERE CarID='1207';
```

This will result.

```
+---------------------+---------------------+
| LCASE(Manufacturer) | UCASE(Manufacturer) |
+---------------------+---------------------+
| toyota              | TOYOTA              |
+---------------------+---------------------+
1 row in set (0.00 sec)
```

7.5.10 Reverse Character Content:

The last string function this section will examine is **REVERSE()**.

7.5.10.1 Syntax:

```
REVERSE(string)
```

This rather obviously reverses the order of the letters. For example the alphabet.

```
mysql> SELECT REVERSE('abcdefghijklmnopqrstuvwxyz');
```

This will result.

```
+---------------------------------------+
| REVERSE('abcdefghijklmnopqrstuvwxyz') |
+---------------------------------------+
| zyxwvutsrqponmlkjihgfedcba            |
+---------------------------------------+
1 row in set (0.00 sec)
```

7.6 Numeric Functions:

Before talking about the specific numeric functions, it is probably worth mentioning that MySQL can perform simple math functions using mathematical operators. Table 7.7 shows them.

Table 7.7 simple math functions using mathematical operators

Operator	Function
+	Add
-	Subtract
*	Multiply
/	Divide

7.6.1 Examples of Simple Math Functions:

```
mysql> SELECT 50+221;
```

This will result.

```
+--------+
| 50+221 |
+--------+
|    271 |
+--------+
1 row in set (0.00 sec)
```

```
mysql> SELECT 50-221;
```

This will result.

```
+--------+
| 50-221 |
+--------+
|   -171 |
+--------+
1 row in set (0.00 sec)
```

```
mysql> SELECT 50*221;
```

This will result.

```
+--------+
| 50*221 |
+--------+
|  11050 |
+--------+
1 row in set (0.00 sec)
```

```
mysql> SELECT 50/221;
```

This will result.

```
+--------+
| 50/221 |
+--------+
|   0.23 |
+--------+
1 row in set (0.00 sec)
```

7.7 More Math Functions:

There are also other functions that serve a more specific math function and we shall have a look at a few of these.

7.7.1 FLOOR():

This reduces any number containing decimals to the lowest whole number.

7.7.1.1 Syntax:

```
SELECT FLOOR(number)
```

7.7.1.2 Example:

```
mysql> SELECT FLOOR(124.845);
```

This will result.

```
+-----------------+
| FLOOR(124.845) |
+-----------------+
|            124 |
+-----------------+
1 row in set (0.00 sec)
```

7.7.2 CEILING():

Raises a number containing decimals to the highest whole number.

7.7.2.1 Syntax:

```
SELECT CEILING(number)
```

7.7.2.2 Example:

```
mysql> SELECT CEILING(124.845);
```

This will result.

```
+-------------------+
| CEILING(124.845) |
+-------------------+
|              125 |
+-------------------+
1 row in set (0.00 sec)
```

7.7.3 ROUND():

This function, as you may have guessed, rounds the figures up or down to the nearest whole number (or to a specified number of decimal places). A main difference is from 5 to 1 would be rounded to zero.

7.7.3.1 Syntax:

```
ROUND(number,[Decimal Places])
```

7.7.3.2 Examples:

```
mysql> SELECT ROUND(124.845,2);
```

This will result.

```
+-------------------+
| ROUND(124.846,2) |
+-------------------+
|           124.84 |
+-------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT ROUND(124.846,2);
```

This will result.

```
+-------------------+
| ROUND(124.846,2) |
+-------------------+
|           124.85 |
+-------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT ROUND(124.846);
```

This will result.

```
+-------------------+
| ROUND(124.846,2)  |
+-------------------+
|           124.85  |
+-------------------+
1 row in set (0.00 sec)
```

'Decimal Places' is optional and omitting it will mean that the figure is rounded to a whole number.

7.7.4 TRUNCATE():

This function, rather than rounding, simply shortens the number to a required decimal place.

7.7.4.1 Syntax:

```
TRUNCATE(number,places)
```

7.7.4.2 Example:

```
mysql> SELECT TRUNCATE(124.896,1);
```

This will result.

```
+---------------------+
| TRUNCATE(124.896,1) |
+---------------------+
|               124.8 |
+---------------------+
1 row in set (0.00 sec)
```

The interesting thing about truncate is that if you specify a negative number for the 'places', it replaces the existing numbers in those places with zeros.

```
mysql> SELECT TRUNCATE(124.896,-2);
```

This will result.

```
+----------------------+
| TRUNCATE(124.896,-2) |
+----------------------+
|                  100 |
+----------------------+
1 row in set (0.00 sec)
```

7.7.5 COUNT():

This counts the number of times a row (or field) is returned.

7.7.5.1 Syntax:

```
COUNT(field)
```

The most common usage for this is just to specify an asterisk as the field to count the number of rows (or in this case cars).

7.7.4.2 Example:

```
mysql> SELECT COUNT(*) as 'Number of Cars' FROM cars;
```

This will result.

```
+-----------------+
| Number of Cars |
+-----------------+
|              5 |
+-----------------+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(Car) as 'Number of Car' FROM cars;
```

This will result.

```
+-----------------+
| Number of Car  |
+-----------------+
|              5 |
+-----------------+
1 row in set (0.01 sec)
```

7.7.6 MIN() and MAX():

These functions are very similar and select the lowest and highest figure respectively from a result set.

7.7.6.1 Syntax:

```
MIN(field)
MAX(field)
```

7.7.6.2 Example:

So a simple example would be to display the least number and most number of CC that any car in the database has.

```
mysql> SELECT MIN(CC), MAX(CC) FROM cars;
```

This will result.

```
+---------+---------+
| MIN(CC) | MAX(CC) |
+---------+---------+
|    1300 |    1600 |
+---------+---------+
1 row in set (0.00 sec)
```

7.7.7 SUM():

The final summary function that we will look at is the SUM() function which adds rows of one field in the results set together.

7.7.7.1 Syntax:

```
SUM(field)
```

7.7.7.2 Example:

So another simple example would be to add the total number of brands of cars having an air condition.

```
mysql> SELECT SUM(AirCon) FROM cars;
```

This will result.

```
+-------------+
| SUM(AirCon) |
+-------------+
|           3 |
+-------------+
1 row in set (0.00 sec)
```

7.8 Control Functions:

The final set of functions will look at are the control functions that allow us a degree of conditionality when returning result sets.

7.8.1 IF():

The IF() function is fairly straight forward and consists of 3 elements. A condition and values for the condition being evaluated either true or false.

7.8.1.1 Syntax:

```
IF(condition,true_value,false_value)
```

7.8.1.2 Example:

So using a simple comparison (is a number greater than 10) to return either 'yup' or 'nope'.

```
mysql> SELECT IF(105>106,'True','False');
```

This will result.

```
+----------------------------+
| IF(105>106,'True','False') |
+----------------------------+
| False                      |
+----------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT IF(105<106,'True','False');
```

This will result.

```
+----------------------------+
| IF(105<106,'True','False') |
+----------------------------+
| True                       |
+----------------------------+
1 row in set (0.00 sec)
```

As well as returned a string value, a number can also be returned. Thus if we wanted to count cars had CC less than 1500.

```
mysql> SELECT IF(CC<1500,1,NULL) FROM cars;
```

This will result.

```
+--------------------+
| IF(CC<1500,1,NULL) |
+--------------------+
|                  1 |
|               NULL |
|                  1 |
|                  1 |
|               NULL |
+--------------------+
5 rows in set (0.00 sec)
```

We can then use COUNT() to give us a total as it ignores NULL values.

```
mysql> SELECT COUNT(IF(CC<1500,1,NULL)) FROM cars;
```

This will result.

```
+---------------------------+
| COUNT(IF(CC<1500,1,NULL)) |
+---------------------------+
|                         3 |
+---------------------------+
1 row in set (0.00 sec)
```

2.8.2 CASE:

Slightly more advanced from IF() is the CASE function that allows for than one comparison to be made. It is slightly different as the actual value is specified first, then a series of comparisons are made for a potential match that then returns a value.

2.8.2.1 Syntax:

```
CASE actual_value WHEN potential_value1 THEN return_value1 WHEN potential_value2 THEN return_value2...etc END
```

7.8.2.2 Example:

Thus if we were to evaluate numeric values and return their string values.

```
mysql> SELECT CASE 2 WHEN 1 THEN 'One' WHEN 2 THEN 'Two' WHEN 3
       THEN 'Three' END;
```

This will result.

```
+-------------------------------------------------------------------+
| CASE 2 WHEN 1 THEN 'One' WHEN 2 THEN 'Two' WHEN 3 THEN 'Three' END |
+-------------------------------------------------------------------+
| Two                                                               |
+-------------------------------------------------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT CASE 1 WHEN 1 THEN 'One' WHEN 2 THEN 'Two' WHEN 3
       THEN 'Three' END;
```

This will result.

```
+-------------------------------------------------------------------+
| CASE 1 WHEN 1 THEN 'One' WHEN 2 THEN 'Two' WHEN 3 THEN 'Three' END |
+-------------------------------------------------------------------+
| One                                                               |
+-------------------------------------------------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT CASE 3 WHEN 1 THEN 'One' WHEN 2 THEN 'Two' WHEN 3
       THEN 'Three' END;
```

This will result.

```
+-------------------------------------------------------------------+
| CASE 3 WHEN 1 THEN 'One' WHEN 2 THEN 'Two' WHEN 3 THEN 'Three' END |
+-------------------------------------------------------------------+
| Three                                                             |
+-------------------------------------------------------------------+
1 row in set (0.00 sec)
```

7.8.3 IFNULL()

The Final function we will look at is IFNULL and unsurprisingly this is a very simple syntax that is similar to IF(). The difference is that that instead of there being TRUE and FALSE return values based on a condition, the original value is returned if it is not NULL and a different new value is returned if it is NULL.

7.8.3.1 Syntax:

```
IFNULL(original_value, new_value)
```

7.8.3.2 Example:

To quickly demonstrate test one query with a NULL value and another with a real value.

```
mysql> SELECT IFNULL(100,'The value is Null');
```

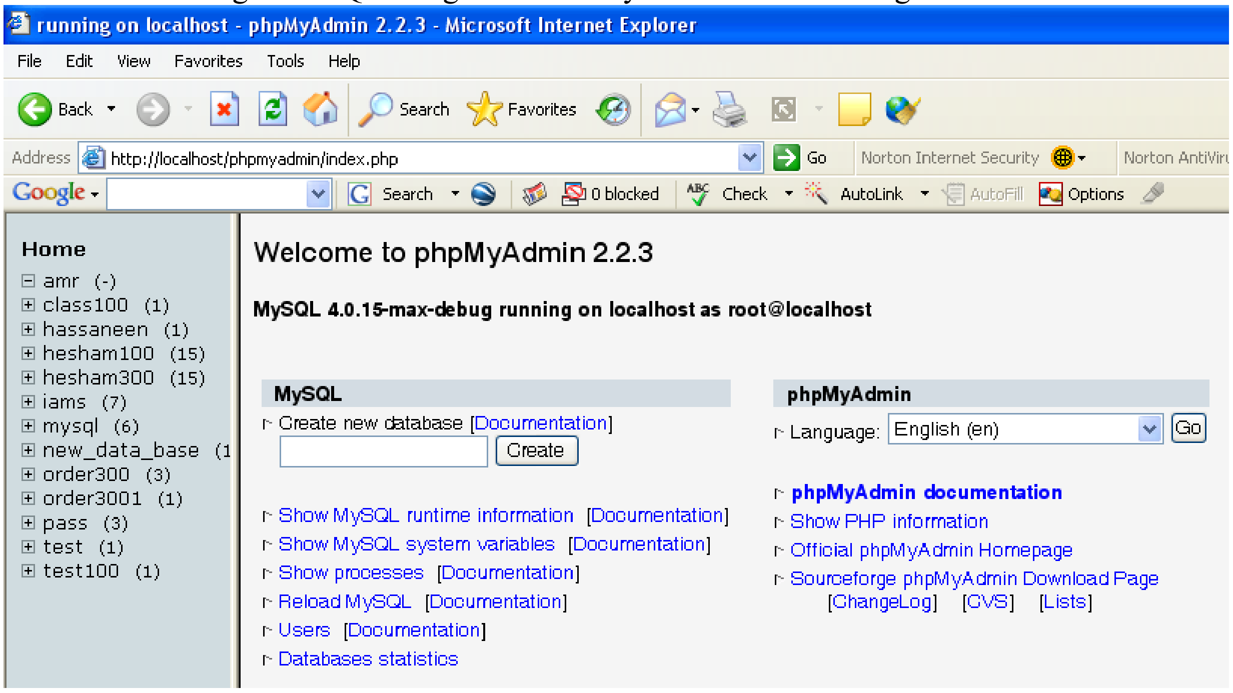
This will result.

```
+---------------------------------+
| IFNULL(100,'The value is Null') |
+---------------------------------+
| 100                             |
+---------------------------------+
1 row in set (0.00 sec)
```

This is particularly useful for converting NULL values to actual numeric zeros.

7.9 PHPMyAdmin:

You can manage MYSQL using the "PHPMyAdmin" shown in figure 7.1


running on localhost - phpMyAdmin 2.2.3 - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites

Address http://localhost/phpmyadmin/index.php Go Norton Internet Security Norton AntiViru

Google Search 0 blocked Check AutoLink AutoFill Options

Home
- amr (-)
- class100 (1)
- hassaneen (1)
- hesham100 (15)
- hesham300 (15)
- iams (7)
- mysql (6)
- new_data_base (1
- order300 (3)
- order3001 (1)
- pass (3)
- test (1)
- test100 (1)

Welcome to phpMyAdmin 2.2.3

MySQL 4.0.15-max-debug running on localhost as root@localhost

MySQL
- Create new database [Documentation] Create
- Show MySQL runtime information [Documentation]
- Show MySQL system variables [Documentation]
- Show processes [Documentation]
- Reload MySQL [Documentation]
- Users [Documentation]
- Databases statistics

phpMyAdmin
- Language: English (en) Go
- phpMyAdmin documentation
- Show PHP information
- Official phpMyAdmin Homepage
- Sourceforge phpMyAdmin Download Page [ChangeLog] [CVS] [Lists]

Figure 7.1 PHPMyAdmin to manage MYSQL.

7.10 Handy MYSQL Commands:

The most used MYSQL commands are shown in table 7.8.

Table 7.8 Handy Commands of MYSQL

Handy MySQL Commands	
Description	**Command**
To login.(from unix shell):	[mysql dir]/bin/mysql -u root -p

List all databases on the sql server:	show databases;
Switch to a database:	use [db name];
To see all the tables in the db:	show tables;
To see database's field formats:	describe [table name];
To delete a db:	drop database [database name];
To delete a table:	drop table [table name];
Show all data in a table:	SELECT * FROM [table name];
Returns the columns and column information pertaining to the designated table:	show columns from [table name];
Show certain selected rows:	SELECT * FROM [table name] WHERE [field name] = "whatever";
Show all records containing the name "Bob" AND the phone number '3444444':	SELECT * FROM [table name] WHERE name = "Bob" AND phone_number = '3444444';
Show all records not containing the name "Bob" AND the phone number '3444444' order by the phone_number field:	SELECT * FROM [table name] WHERE name != "Bob" AND phone_number = '3444444' order by phone_number;
Show all records starting with the letters 'bob' AND the phone number '3444444':	SELECT * FROM [table name] WHERE name like "Bob%" AND phone_number = '3444444';
Join tables on common columns	select lookup.illustrationid, lookup.personid,person.birthday from lookup left join person on lookup.personid=person.personid=statement to join birthday in person table with primary illustration id;
Switch to the mysql db. Create a new user:	INSERT INTO [table name] (Host,User,Password) VALUES('%','user',PASSWORD('password'));
Change a users password.(from unix shell):	[mysql dir]/bin/mysqladmin -u root -h hostname.blah.org -p password 'new-password'
Switch to mysql db.Give user privileges for a db:	INSERT INTO [table name] (Host,Db,User,Select_priv,Insert_priv,Update_priv,Delete_priv,Create_priv,Drop_priv) VALUES ('%','db','user','Y','Y','Y','Y','Y','N');
To update info already in a table:	UPDATE [table name] SET Select_priv = 'Y',Insert_priv = 'Y',Update_priv = 'Y' where [field name] = 'user';
Delete a row(s) from a table:	DELETE from [table name] where [field name] = 'whatever';
Update database permissions/privileges:	FLUSH PRIVILEGES;
Delete a column:	alter table [table name] drop column [column name];
Add a new column to db:	alter table [table name] add column [new column name] varchar (20);
Change column name:	alter table [table name] change [old column name] [new column name] varchar (50);
Make a unique column so you get no dupes:	alter table [table name] add unique ([column name]);
Make a column bigger:	alter table [table name] modify [column name] VARCHAR(3);
Delete unique from table:	alter table [table name] drop index [colmn name];

Load a CSV file into a table:	`LOAD DATA INFILE '/tmp/filename.csv' replace INTO TABLE [table name] FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' (field1,field2,field3);`
Dump all databases for backup.Backup file is sql commands to recreate all db's.	`[mysql dir]/bin/mysqldump --user=root --password=blah --all-databases >/tmp/sql-01_backup.sql`
Create Table Example 1	`CREATE TABLE [table name] (firstname VARCHAR(20), middleinitial VARCHAR(3), lastname VARCHAR(35),suffix VARCHAR(3), officeid VARCHAR(10),userid VARCHAR(15),username VARCHAR(8),email VARCHAR(35),phone VARCHAR(25), groups VARCHAR(15),datestamp DATE,timestamp time,pgpemail VARCHAR(255));`
Create Table Example 2	`create table [table name] (personid int(50) not null auto_increment primary key,firstname varchar(35),middlename varchar(50),lastname varchar(50) default 'bato');`

7.11 A Sample of Questions:

Part 1: ....

Fill in the circle that represents the correct choice for each question in the given answer sheet (more than one choice for any question would be considered wrong answer).

An example of a correct answer:

a	b	c	d
●	○	○	○

Examples of wrong answers:

a	b	c	d
●	●	○	○

a	b	c	d
○	✓	○	○

a	b	c	d
○	◎	○	○

1- To find out the details of fields of any table use the command:
   a- show   b- select.
   c- describe.   d- None of the above

2- To insert new column into an existing table use the command
   a- create   b- update
   c- load   d- alter

3- When using the command

```
mysql> SELECT IFNULL(100,'The value is Null');
```

The result would be
   a- error   b- displays Null all the time
   c- displays 100 for all values   d- displays 100 for Null values

Part 2 : . . . .

Fill in the circle that represents the correct choice for each question in the given answer sheet (more than one choice for any question would be considered wrong answer).

1) The command mysql> **SELECT COUNT(IF(CC<1500,1,NULL)) FROM cars;** will display how many cars have the CC less than 1500.
   a. True
   b. False

2) The result of this command:
   mysql> **SELECT IF(105<106, 'False', 'True');** would be:
   a. True
   b. False

Part 3: ....
This part consists of fill in The Blank questions. Given below a table of words to choose from. On the answer sheet, put the number of the appropriate word in the space available for that question.

1 alter	2 add	3 if	4 Authorization	5 exit
6 switch	7 do	8 assign	9 FLUSH PRIVILEGES;	10 delete

1. . . . .1 . table [table name] modify [column name] VARCHAR(10); is used to change the column size and type.
2. alter table [table name] add column [new column name] varchar (20); is use to . . . . .2 . a new table.
3. To Update database permissions/privileges: use. . . 4.
4. alter table [table name] drop column [column name]; is used . . . 10 a column from a table.