Lab 7: SQL Functions

The learning objectives of this lab are to

- Learn about selected MySQL date and time functions

- Be able to perform string manipulations

- Utilise single row numeric functions

- Perform conversions between data types

There are many types of SQL functions, such as arithmetic, trigonometric, string, date, and time functions. Lab 7 will cover a selection of these SQL functions that are implemented in MySQL in detail. Functions always use a numerical, date, or string value. The value may be part of the command itself (a constant or literal) or it may be an attribute located in a table. Therefore, a function may appear anywhere in a SQL statement where a value or an attribute can be used.

7.1 Date and Time Functions

In MySQL there are a number of useful date and time functions. However, first it is important to briefly look at the main date and time types are available to MySQL. These are shown in the table below:

Table 7.1 MySQL Date and Time data types

DATETIME	YYYY-MM-DD HH:MM:SS
DATE	YYYY-MM-DD
TIMESTAMP	YYYYMMDDHHSSMM
TIME	HH:MM:SS

YEAR	YYYY

As you can see from Table 7.1, the DATE type is stored in a special internal format that includes just the year, month and day whilst the DATETIME data type also stores the hours, minutes, and seconds. If you try to enter a date in a format other than the Year-Month-Day format then it might work, but it won't be storing them as you expect!

Task 7.1 Enter the following query and examine how the date is displayed.

SELECT   DISTINCT(SALE_DATE)

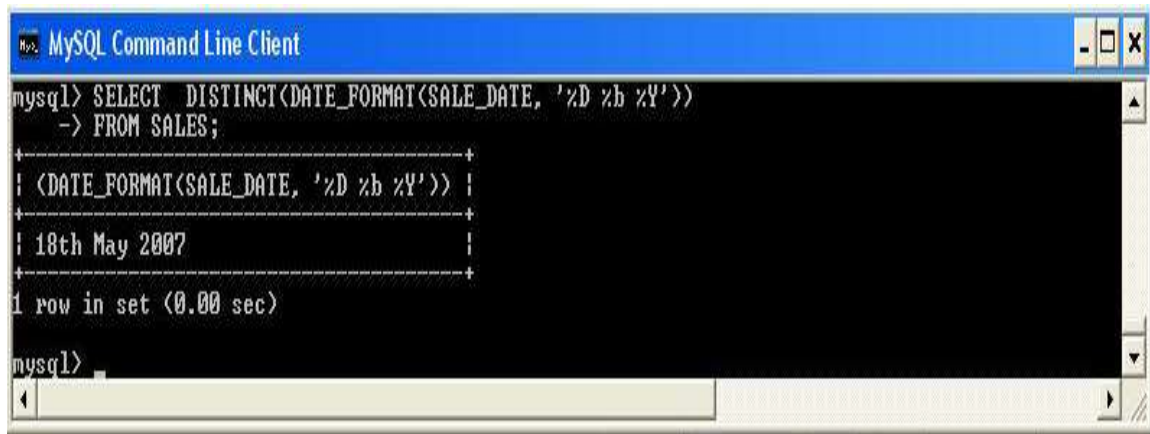FROM   SALES;

It is possible to change the format of the date using the DATE_FORMAT() function. The syntax of this function is

DATE_FORMAT(date,format)

The function formats the date value according to the format string.

For example, the following query formats the date as 18th May 2007 using ' date specifiers' as shown in Figure 55.

SELECT DISTINCT(DATE_FORMAT(SALE_DATE, '%D %b %Y'))

FROM SALES;

Figure 55 Formatting Dates in MySQL

Table 7.2 taken directly from the MySQL Manual 5.0 shows a complete list of specifiers

that can be used in the *format* string.

Specifier	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, ...)
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)
%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00..53), where Sunday is the first day of the week
%u	Week (00..53), where Monday is the first day of the week
%V	Week (01..53), where Sunday is the first day of the week; used with %X

%V	Week (01..53), where Monday is the first day of the week; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric (two digits)
%%	A literal '%' character
%x	x, for any 'x' not listed above

Task 7.2 Using the date specifiers in Table 7.2, modify the query shown in Figure 55 to display the date in the format 'Fri – 18 – 5 – 07'.

You will now explore some of the main MySQL date / time functions.

CURRENT DATE and CURRENT TIME

The CURRENT_DATE function returns today's date while the CURRENT_TIME function returns the current time.

Task 7.3 Enter the following query to display today's date and time. Notice that in MySQL the functions are called using the SELECT statement but no FROM clause is needed.

mysql> SELECT CURRENT_DATE(), CURRENT_TIME();

> ***Note***
>
> CURRENT_TIME and CURRENT_DATE are synonyms for CURTIME() and
>
> CURDATE respectively.

The output for this query is shown in Figure 56.

Figure 56 Displaying the current date and time.

MONTH, DAYOFMONTH and YEAR

MySQL provides functions for extracting the month, day or year from any given date.
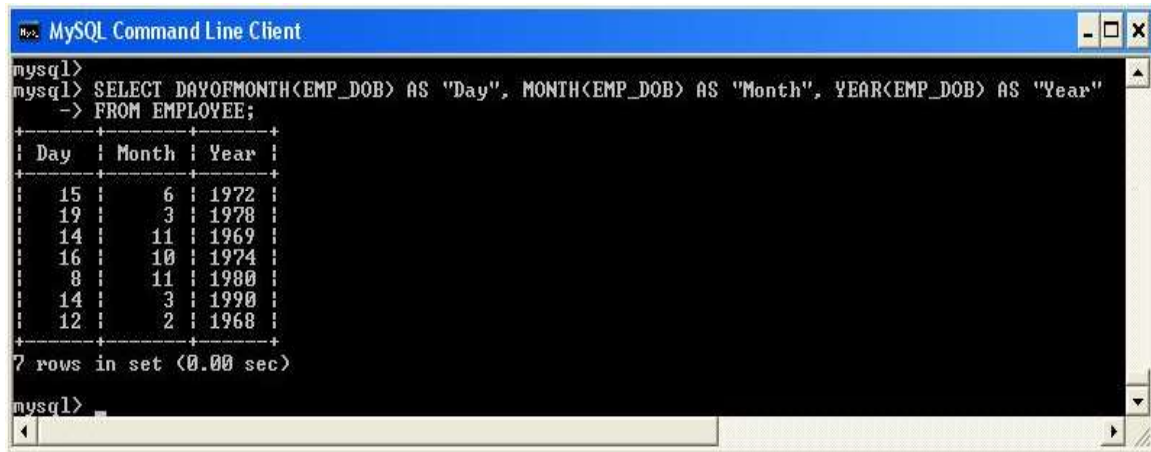
The syntax of each function is as follows:

DAYOFMONTH(date) returns the day of the month for date, in the range 0 to 31.

MONTH(date) returns the month for date, in the range 0 to 12.

YEAR(date) returns the year for date, in the range 1000 to 9999, or 0 for the "zero" date.

The following query shows how these three functions can be used to display different

parts of an employee's date of birth. The output of this query is shown in Figure 57.

SELECT DAYOFMONTH(EMP_DOB) AS "Day", MONTH(EMP_DOB) AS "Month",

YEAR(EMP_DOB) AS "Year"

FROM EMPLOYEE;

Figure 57 Using the MONTH, DAYOFMONTH and YEAR functions.

Task 7.3 Write a query that displays all employees who were born in November. Your

output should match that shown in Figure 58.

Figure 58 Output for Task 7.3.

DATEDIFF

The DATEDIFF function subtracts two dates and returns a value in days from one date to the other. The following example calculates the number of days between the 1st January 2008 and the 25th December 2008.

SELECT DATEDIFF('2008-12-25','2008-01-01');

Task 7.4 Enter the query above and see how many days it is until the 25th December. Then modify the query to see how many days it is from today's date until 25th December 2009.

DATE_ADD and DATE_SUB

The DATE_ADD and DATE_SUB functions both perform date arithmetic and allow you to either add or subtract two dates from one another. The syntax of these functions is:

DATE_ADD(date,INTERVAL expr unit)

DATE_SUB(date,INTERVAL expr unit)

Where expr is an expression specifying the interval value to be added or subtracted from the starting date and unit is a keyword indicating the units in which the expression should be interpreted.

For example, the following query adds 11 months to the date 1st January 2008 to display a new date of 1st December 2008. The output for this query is shown in Figure 59.

SELECT ADDDATE('2008-01-01', INTERVAL 11 MONTH);

Figure 59 Adding months to a date

A full list of the different interval types can be found in the MySQL Reference Manual 5.0.

Task 7.6 Enter the following query which lists the hire dates of all employees along with the date of their first work appraisal (one year from the hiredate). Check that the output is correct.

SELECT EMP_LNAME, EMP_FNAME, EMP_HIRE_DATE,

ADDDATE(EMP_HIRE_DATE, INTERVAL 12 MONTH)AS "FIRST APPRAISAL"

FROM EMPLOYEE;

LAST_DAY

The function LAST_DAY returns the date of the last day of the month given in a date.

The syntax is

LAST_DAY(date_value).

Task 7.7 Enter the following query which lists all sales transactions that were made in the last 20 days of a month:

SELECT *

FROM SALES

WHERE SALE_DATE >= LAST_DAY(SALE_DATE)-20;

7.2 Numeric Functions

In this section, you will learn about MySQL single row numeric functions. Numeric functions take one numeric parameter and return one value. A description of the functions you will explore in this lab can be found in Table 4.

> *Note*
>
> Do not confuse the SQL aggregate functions you saw in the previous chapter with the numeric functions in this section. The first group operates over a set of values (multiple rows—hence, the name *aggregate functions*), while the numeric functions covered here operate over a single row.

Table 4 Selected Numeric Functions
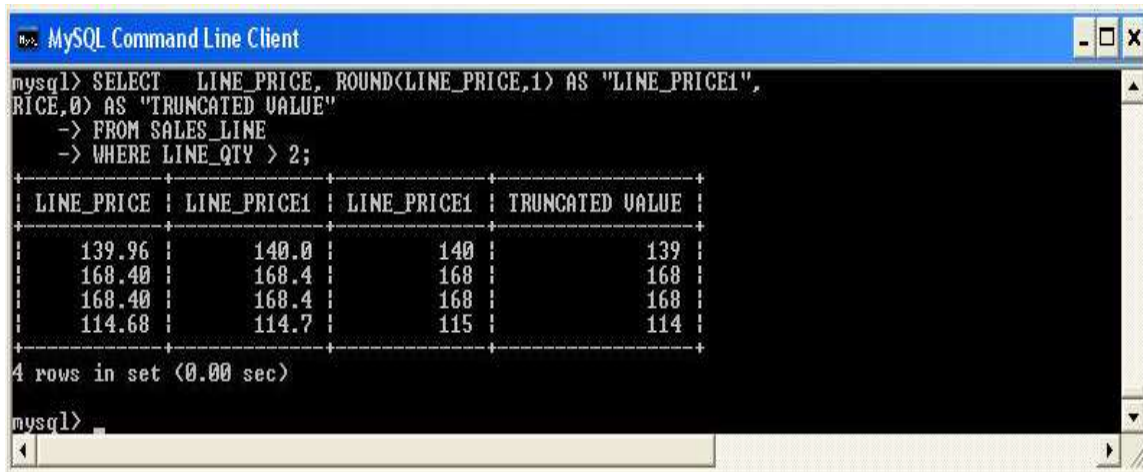
Function	Description
ABS	Returns the absolute value of a number Syntax: ABS(numeric_value)
ROUND	Rounds a value to a specified precision (number of digits)

	Syntax: ROUND(numeric_value, p) where p = precision
TRUNCATE	Truncates a value to a specified precision (number of digits)
	Syntax: TRUNC(numeric_value, p) where p = precision
MOD	Returns the remainder of division.
	Syntax MOD(m.n) where m is divided by n.

The following example displays the individual LINE_PRICE from the sales line table,

rounded to one and zero places and truncated where the quantity of tickets purchased on

that line is greater than 2.

SELECT   LINE_PRICE, ROUND(LINE_PRICE,1) AS "LINE_PRICE1",

                ROUND(LINE_PRICE,0) AS "LINE_PRICE1",

TRUNCATE(LINE_PRICE,0) AS "TRUNCATED VALUE"

FROM SALES_LINE

WHERE LINE_QTY > 2;

The output for this query can be seen in Figure 60.

Figure 60 Example of ROUND and TRUNC

Task 7.8 Enter the following query and execute it. Can you explain the results of this query?

SELECT TRANSACTION_NO, LINE_PRICE, MOD(LINE_PRICE, 10)

FROM SALES_LINE

WHERE LINE_QTY > 2;

7.3 String Functions

String manipulation functions are amongst the most-used functions in programming.

Table 5 shows a subset of the most useful string manipulation functions in MySQL.

Table 5 Selected MySQL string functions.
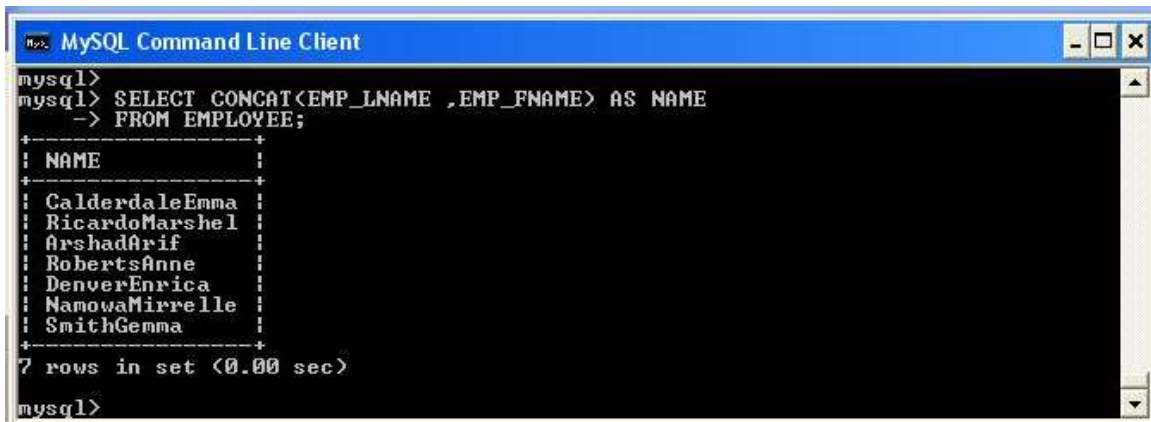
Function	Description
CONCAT	Concatenates data from two different character columns and returns a single column. Syntax: CONCAT(strg_value, strg_value)
UPPER/LOWER	Returns a string in all capital or all lowercase letters Syntax: UPPER(strg_value) , LOWER(strg_value)
SUBSTR	Returns a substring or part of a given string parameter Syntax: SUBSTR(strg_value, p, l) where p = start position and l = length of characters
LENGTH	Returns the number of characters in a string value Syntax: LENGTH(strg_value)

We will now look at examples of some of these string functions.

CONCAT

The following query illustrates the CONCAT function. It lists all employee first and last

names concatenated together. The output for this query can be seen in Figure 61.

SELECT CONCAT(EMP_LNAME ,EMP_FNAME) AS NAME

FROM EMPLOYEE;

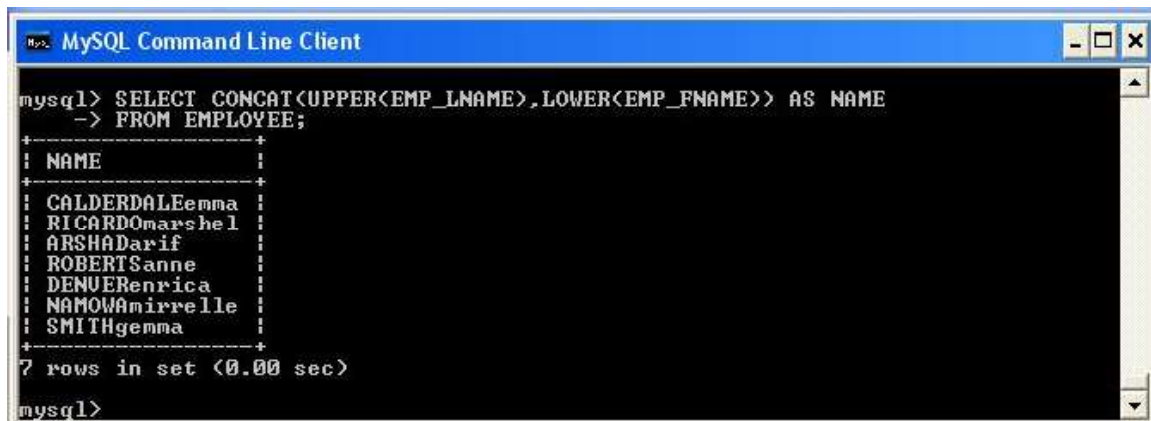**Figure 61 Concatenation of employee's first and last names**

UPPER/LOWER

The following query lists all employee last names in all capital letters and all first names

in all lowercase letters. The output for the query is shown in Figure 62.

SELECT CONCAT(UPPER(EMP_LNAME),LOWER(EMP_FNAME)) AS NAME

FROM EMPLOYEE;
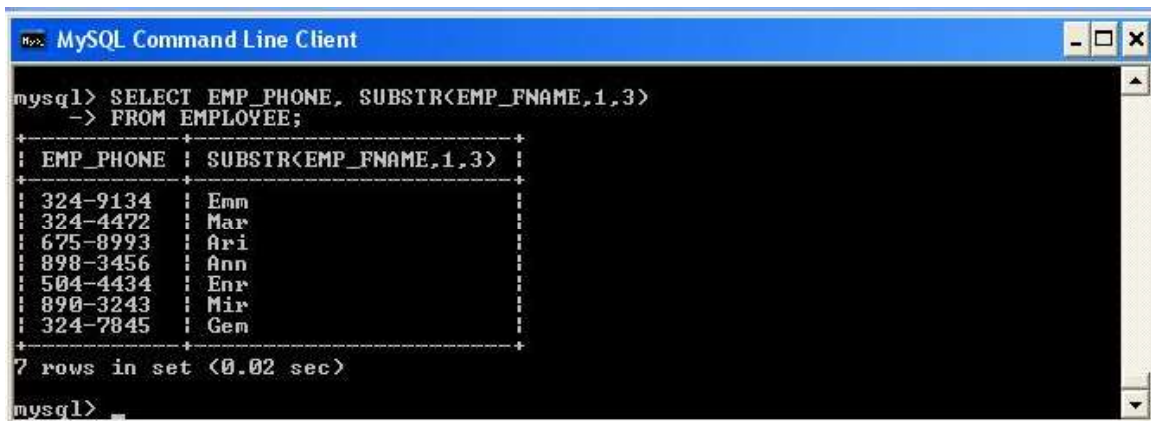
Figure 62 Displaying upper and lower case employee names.

SUBSTR

The following example lists the first three characters of all the employees' first name.

The output of this query is shown in Figure 63.

SELECT EMP_PHONE, SUBSTR(EMP_FNAME,1,3)

FROM EMPLOYEE;

Figure 63 Displaying the first 3 characters of the employees first name

Task 7.10 Write a query which generates a list of employee user IDs, using the first day of the month they were born and the first six characters of last name in UPPER case.

Your query should return the results shown in Figure 64.
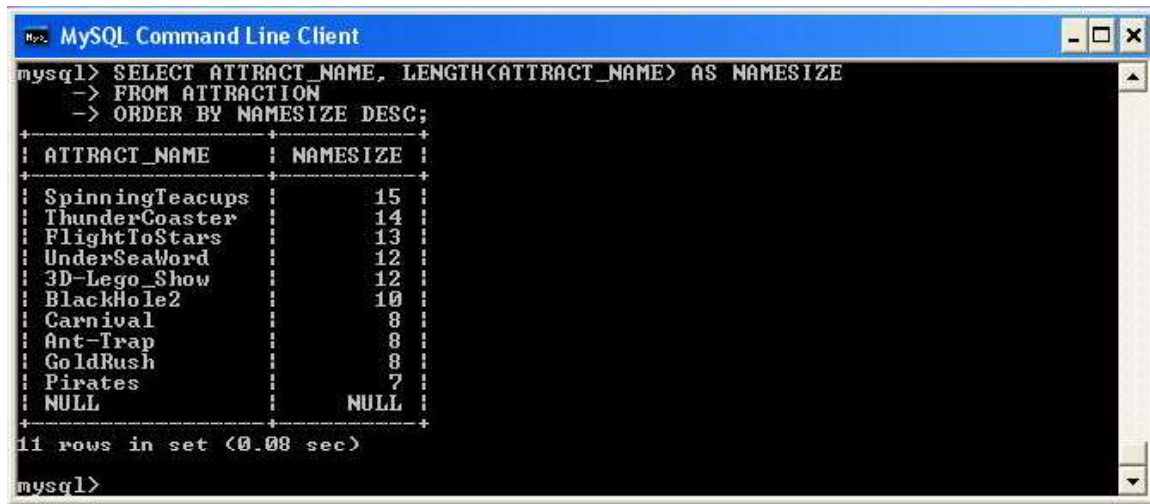

Figure 64 Results for Task 7.10.

LENGTH

The following example lists all attraction names and the length of their names; ordered

descended by attraction name length. The output of this query is shown in Figure 65.

SELECT ATTRACT_NAME, LENGTH(ATTRACT_NAME) AS NAMESIZE

FROM ATTRACTION

ORDER BY NAMESIZE DESC;

Figure 65 Displaying the length of attraction names.

7.4 Conversion Functions

Conversion functions allow you to take a value of a given data type and convert it to the

equivalent value in another data type. In MySQL, some conversions occur implicitly. For

example, MySQL automatically converts numbers to strings when needed, and vice

versa.

So if you enter the following query:

SELECT 10 + '10'

MySQL would give you an answer of 20 as it would automatically convert the string

containing '10' into the number 10 (see figure 66).

If you want to explicitly convert a number to a string then you can use either the **CAST**

or CONCAT function. However MySQL 5.0 recommends only the CAST function is

used. Let's look at an example. The following query produces the output shown in

Figure 66.

SELECT 10, CAST(10 AS CHAR);

Figure 66 Example of type conversions

> **Note**
>
> The MySQL Reference Manual 5.0 provides a set of rules that allow us to determine how the coversion will occur when using the CONVERT function on different data types.

IFNULL

The IFNULL function lets you substitute a value when a null value is encountered in the results of a query. The syntax is:

IFNULL(expr1,expr2)

If expr1 is not NULL, IFNULL() returns expr1; otherwise it returns expr2. It is equivalent to Oracle's NVL function. It is useful for avoiding errors caused by incorrect calculation when one of the arguments is null.

Task 7.11 Load and run the script sales_copy.sql which accompanies this lab guide. DESCRIBE the structure of the SALES_COPY table and examine the lack of constraints on this table. Write a query to view all the rows and notice that in some rows no values have been entered for LINE_QTY or LINE_PRICE. (In these instances these rows have NULL values.) Next, enter the following query which displays to the screen the Total of the LINE_QTY * LINE_PRICE. Notice that this query does not use the IFNULL function and in two rows the calculation can not be made.

SELECT TRANSACTION_NO, LINE_NO, LINE_QTY, ITEM_PRICE,
LINE_QTY*ITEM_PRICE AS "TOTAL SALES PER LINE"

FROM SALES_COPY;

Next run the following version of the query which uses the IFNULL function and notice that the calculation has been achieved for all rows.

SELECT TRANSACTION_NO, LINE_NO,
IFNULL(LINE_QTY,0),ITEM_PRICE,(IFNULL(LINE_QTY,0))*ITEM_PRICE AS
"TOTAL SALES PER LINE"

FROM SALES_COPY;

The results of running both these queries can be seen in Figure 67.

Figure 67 Illustration of the IFNULL function.

CASE

The CASE function compares an attribute or expression with a series of values and returns an associated value or a default value if no match is found. There are two versions of the CASE function. The syntax of each is shown below.
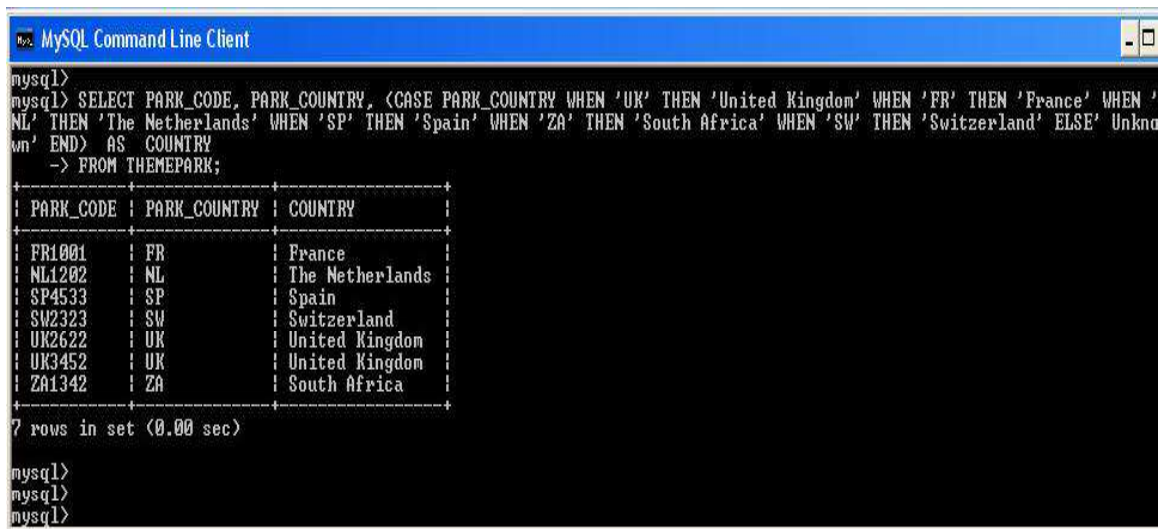
CASE value WHEN [compare_value] THEN result [WHEN [compare_value] THEN result ...] [ELSE result] END

CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...] [ELSE result] END

The first version returns the result where value=compare_value. The second version returns the result for the first condition that is true. If there was no matching result value, the result after ELSE is returned, or NULL if there is no ELSE part.

Let's now look at the following example, which compares the country code in the PARK_COUNTRY field and decodes it into the name of the country. If there is no match, it returns the value 'Unknown'. The output is shown in Figure 68.

SELECT PARK_CODE, PARK_COUNTRY, (CASE PARK_COUNTRY WHEN 'UK' THEN 'United Kingdom' WHEN 'FR' THEN 'France' WHEN 'NL' THEN 'The Netherlands' WHEN 'SP' THEN 'Spain' WHEN 'ZA' THEN 'South Africa' WHEN 'SW' THEN 'Switzerland' ELSE' Unknown' END) AS COUNTRY

FROM THEMEPARK;

Figure 68 Displaying the names of countries using the DECODE function.

It is worth noting that the above decode statement is equivalent to the following IF-

THEN-ELSE statement:

```
IF PARK_COUNTRY = 'UK' THEN
    result := 'United Kingdom';
ELSIF PARK_COUNTRY = 'FR' THEN
   result := 'FRANCE';
ELSIF PARK_COUNTRY = 'NL' THEN
   result := 'The Netherlands';
ELSIF PARK_COUNTRY = 'SP' THEN
   result := 'Spain';
ELSIF PARK_COUNTRY = 'ZA' THEN
   result := 'South Africa';
ELSIF PARK_COUNTRY = 'SW' THEN
   result := 'Switzerland';
ELSE
   result := 'Unknown;
END IF;
```

7.5 Exercises

E7.1 Write a query which lists the names and dates of births of all employees born on the

14th day of the month.

E7.2 Write a query which lists the approximate age of the employees on the company's

tenth anniversary date (11/25/2008).

E7.3 Write a query which generates a list of employee user passwords, using the first

three digits of their phone number, and the first two characters of first name in lower

case. Label the column USER_PASSWORD;

E7.4 Write a query which displays the last date a ticket was purchased in all Theme Parks. You should also display the Theme Park name. Print the date in the format 12th January 2007.