

Lab 6: Views and SQL Functions

The learning objectives of this lab are to

- Create a simple view
- Manage database constraints in views using **WITH CHECK OPTION**
- Learn about selected MySQL date and time functions
- Be able to perform string manipulations
- Utilize single row numeric functions
- Perform conversions between data types

6.1 Views

A **view** is a virtual table based on a **SELECT** query. The query can contain columns, computed columns, aliases, and aggregate functions from one or more tables. The tables on which the view is based are called **base tables**. You can create a view by using the **CREATE VIEW** command:

```
CREATE VIEW viewname AS SELECT query
```

The **CREATE VIEW** statement is a data definition command that stores the subquery specification—the **SELECT** statement used to generate the virtual table—in the data dictionary. For example, to create a view to display the employees first and last name (**EMP_FNAME** and **EMP_LNAME**), the attraction number (**ATTRACT_NO**) and the date worked, you would do so as follows:

```
CREATE VIEW EMP_WORKED AS  
  
SELECT E.EMP_LNAME, E.EMP_FNAME, H.ATTRACT_NO,H.DATE_WORKED  
  
FROM employee E JOIN hours H  
  
ON E.EMP_NUM = H.EMP_NUM;
```

To display the contents of this view you would type

SELECT * FROM EMP_WORKED;

The created view can be seen in figure 78.

```
mysql> CREATE VIEW EMP_WORKED AS
-> SELECT E.EMP_LNAME,E.EMP_FNAME,H.ATTRACT_NO,H.DATE_WORKED FROM
-> employee E JOIN hours H ON E.EMP_NUM = H.EMP_NUM;
Query OK, 0 rows affected (0.11 sec)

mysql> DESCRIBE EMP_WORKED;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| EMP_LNAME  | varchar(15)   | NO   |     | NULL    |       |
| EMP_FNAME  | varchar(15)   | NO   |     | NULL    |       |
| ATTRACT_NO | decimal(10,0) | NO   |     | NULL    |       |
| DATE_WORKED| date          | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT * FROM EMP_WORKED;
+-----+-----+-----+-----+
| EMP_LNAME | EMP_FNAME | ATTRACT_NO | DATE_WORKED |
+-----+-----+-----+-----+
| Calderdale | Emma      | 10034      | 2007-05-18  |
| Calderdale | Emma      | 10034      | 2007-05-20  |
| Ricardo    | Marshal   | 10034      | 2007-05-18  |
| Arshad     | Arif      | 30012      | 2007-05-23  |
| Arshad     | Arif      | 30044      | 2007-05-21  |
| Arshad     | Arif      | 30044      | 2007-05-22  |
| Denver     | Enrica    | 30011      | 2007-05-21  |
| Denver     | Enrica    | 30012      | 2007-05-22  |
| Namowa     | Mirrelle  | 10078      | 2007-05-18  |
| Namowa     | Mirrelle  | 10098      | 2007-05-18  |
| Namowa     | Mirrelle  | 10098      | 2007-05-19  |
+-----+-----+-----+-----+
11 rows in set (0.02 sec)
```

Figure 78 Creating the EMP_WORKED view.

Task 6.1 Create the EMP_WORKED view.

Relational view has several special characteristics. These are:

- You can use the name of a view anywhere a table name is expected in a SQL statement
- Views are dynamically updated. That is, the view is re-created on demand each time it is invoked.
- Views provide a level of security in the database because the view can restrict users to only specified columns and specified rows in a table

To remove the view EMP_WORKED you could issue the following command
DROP VIEW EMP_WORKED;

6.2 Views – using the WITH CHECK OPTION

It is possible to perform referential integrity constraints through the use of a view so that database constraints can be enforced. The following view DISPLAYS employees who work in Theme Park FR1001 using the WITH CHECK OPTION clause. This clause ensures that INSERTs and UPDATEs cannot be performed on any rows that the view has not selected. The results of creating this view can be seen in Figure 79.

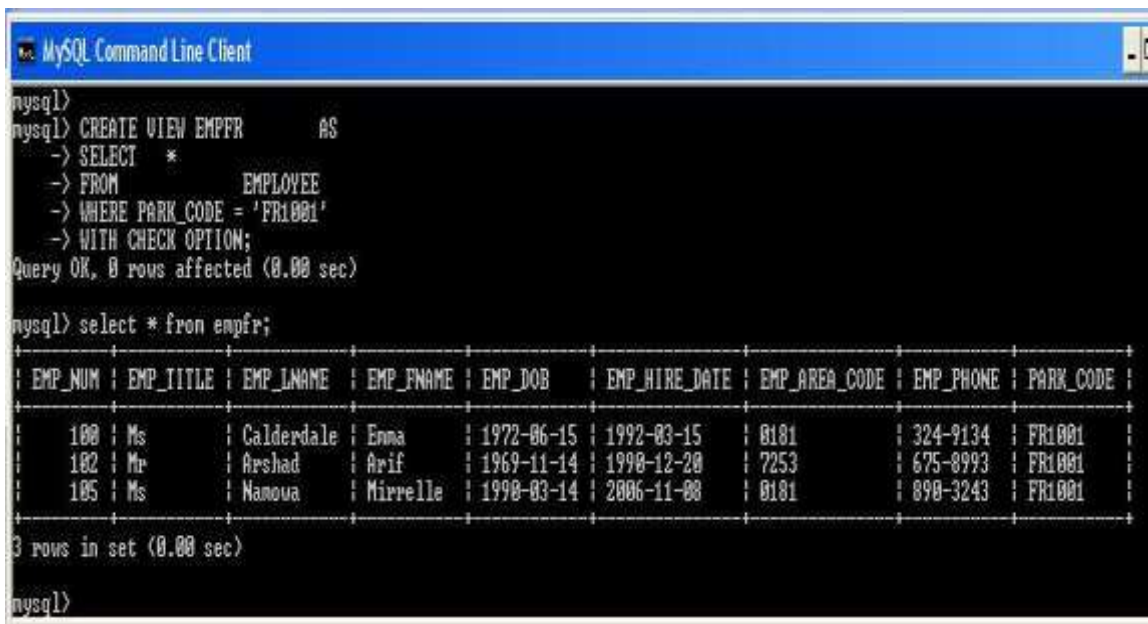
```
CREATE VIEW EMPFR      AS

SELECT      *

FROM        EMPLOYEE

WHERE PARK_CODE = 'FR1001'

WITH CHECK OPTION;
```



```
mysql>
mysql> CREATE VIEW EMPFR      AS
  -> SELECT      *
  -> FROM        EMPLOYEE
  -> WHERE PARK_CODE = 'FR1001'
  -> WITH CHECK OPTION;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from empfr;
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| EMP_NUM | EMP_TITLE | EMP_LNAME | EMP_FNAME | EMP_DOB | EMP_HIRE_DATE | EMP_AREA_CODE | EMP_PHONE | PARK_CODE |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| 100 | Ms | Calderdale | Emma | 1972-06-15 | 1992-03-15 | 0181 | 324-9134 | FR1001 |
| 102 | Mr | Arshad | Arif | 1969-11-14 | 1990-12-20 | 7253 | 675-8993 | FR1001 |
| 105 | Ms | Namova | Mirrelle | 1998-03-14 | 2006-11-08 | 0181 | 890-3243 | FR1001 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Figure 79 Creating the EMPFR view

So for example if employee ‘Emma Caulderdale’ was to leave the park and move to park ‘UK3452’, we would want to update her information with the following query:

UPDATE EMPFR

SET PARK_CODE = 'UK3452'

WHERE EMP_NUM = 100;

However running this update gives the errors shown in Figure 80. This is because if the update was to occur, the view would no longer be able to see this employee.

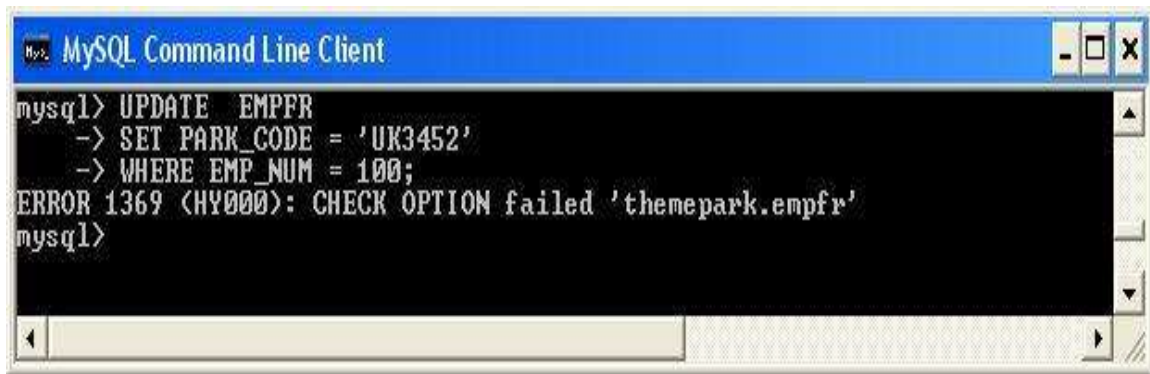


Figure 80 Creating the EMPFR view

Advantages of Views:

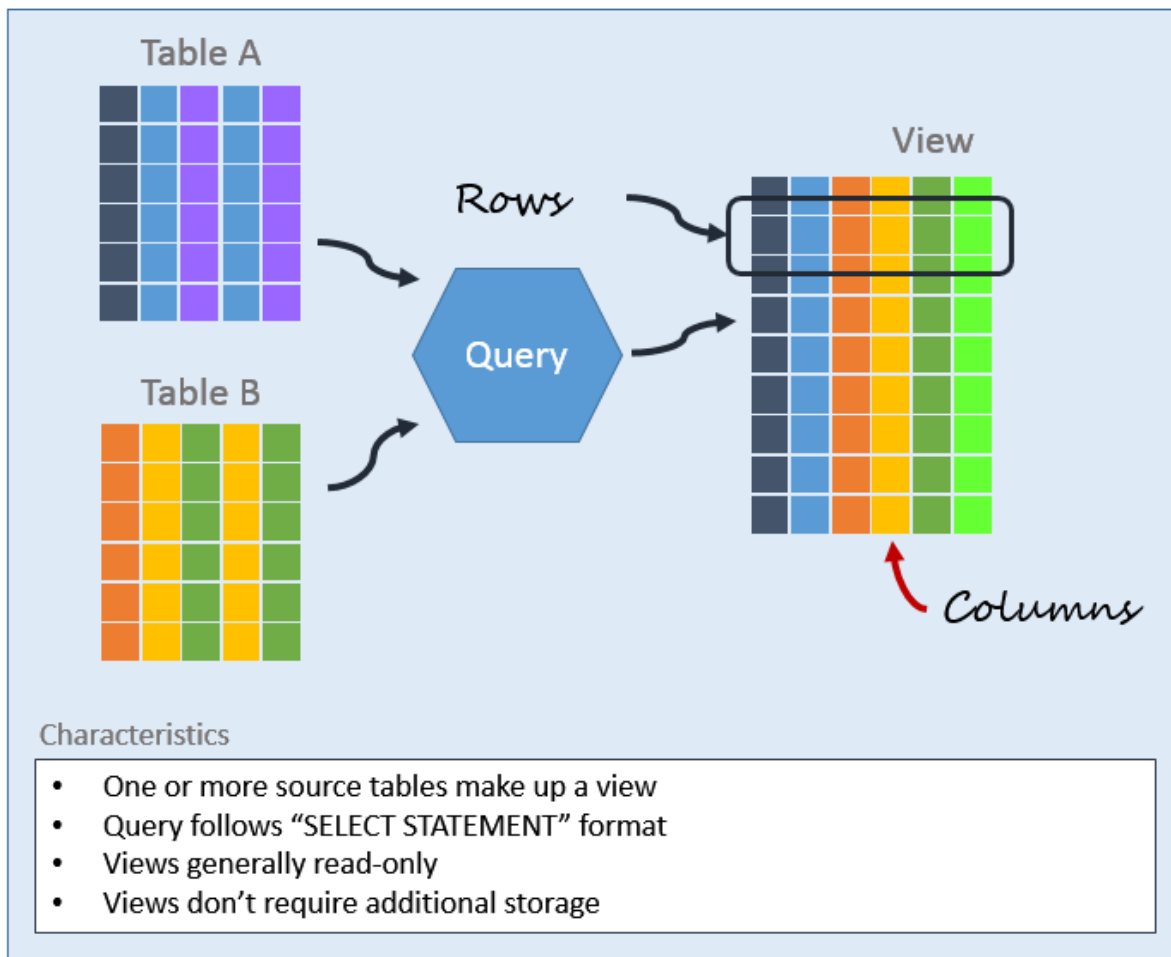
I. Simplify complex Query:

Views help simplify complex queries. If you have any frequently used complex query, you can create a view based on it so that you can reference to the view by using a simple **SELECT** statement instead of typing the query all over again.

II. Add extra security layers:

A table may expose a lot of data including sensitive data such as personal and banking information. By using views and privileges, you can limit which data users can access by exposing only the necessary data to them.

Anatomy of a View



SQL Functions:

There are many types of SQL functions, such as arithmetic, trigonometric, string, date, and time functions. Lab 6 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.

6.3 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 available to MySQL. These are shown in the table below:

Table 6.1 MySQL Date and Time data types

DATETIME	YYYY-MM-DD HH:MM:SS
DATE	YYYY-MM-DD
TIMESTAMP	YYMMDDHHSSMM
TIME	HH:MM:SS
YEAR	YYYY

As you can see from Table 6.1, 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 6.1 Enter the following query and examine how the date is displayed.

```
SELECT    DISTINCT(SALE_DATE )
FROM      SALES;
```

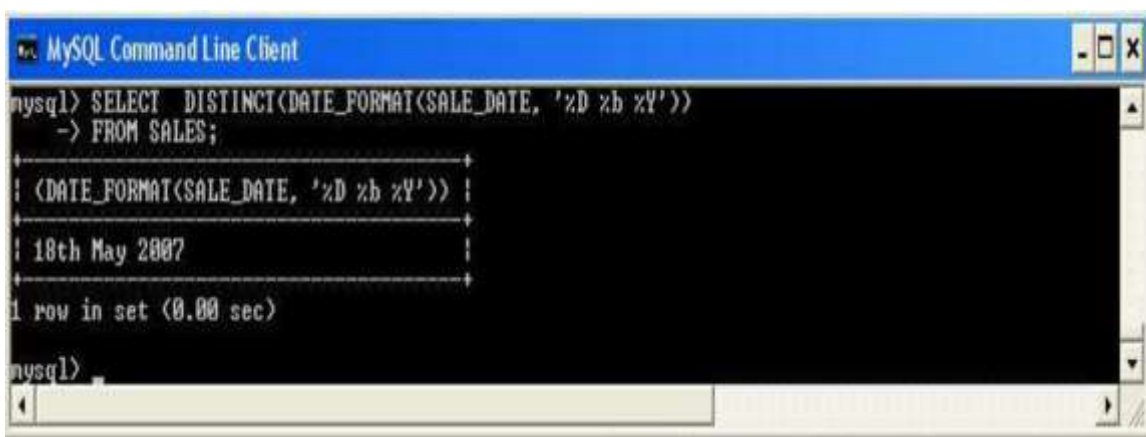
It is possible to change the format of the date using the DATE_FORMAT() function. The syntax of this function is

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;
```



```
mysql> SELECT DISTINCT(DATE_FORMAT(SALE_DATE, '%D %b %Y'))  
-> FROM SALES;  
  
+-----+  
| (DATE_FORMAT(SALE_DATE, '%D %b %Y')) |  
+-----+  
| 18th May 2007                         |  
+-----+  
1 row in set (0.00 sec)  
  
mysql>
```

Figure 55 Formatting Dates in MySQL

Table 6.2 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

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.

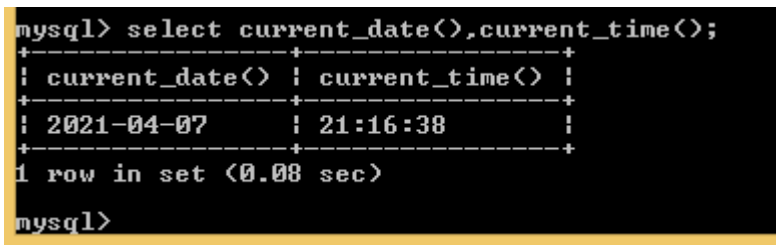
Task 6.3 Enter the following query to display today's date and time. Notice that in MySQL the functions are called using the SELECT statement but no FROM clause is needed.

```
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.



```
mysql> select current_date(), current_time();
+-----+-----+
| current_date() | current_time() |
+-----+-----+
| 2021-04-07     | 21:16:38      |
+-----+-----+
1 row in set (0.08 sec)
mysql>
```

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.

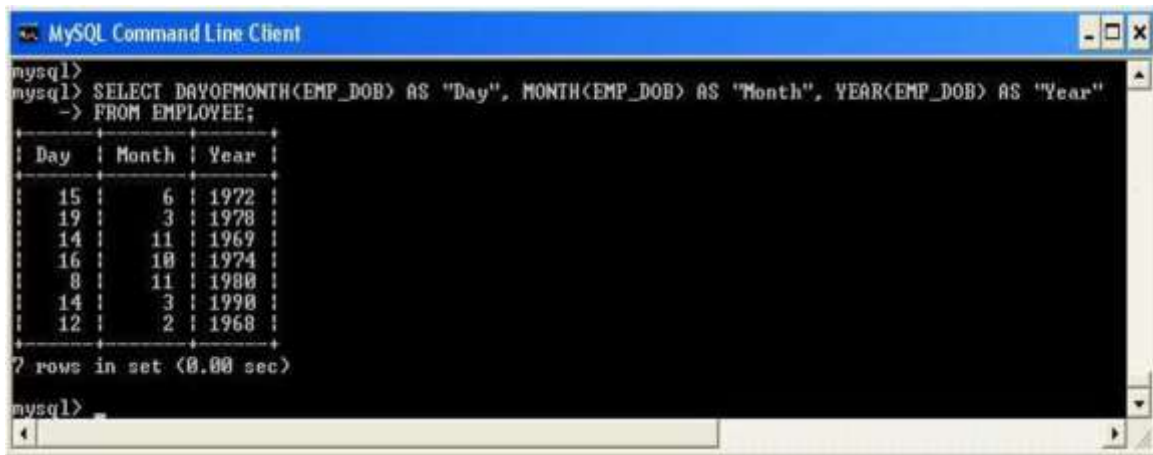
YEAR(date) returns the year for date, in the range 1000 to 9999, or 0 for the “zero” date.

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;

```



```

mysql>
mysql> SELECT DAYOFMONTH(EMP_DOB) AS "Day", MONTH(EMP_DOB) AS "Month", YEAR(EMP_DOB) AS "Year"
-> FROM EMPLOYEE;

```

Day	Month	Year
15	6	1972
19	3	1978
14	11	1969
16	10	1974
8	11	1980
14	3	1990
12	2	1968

```

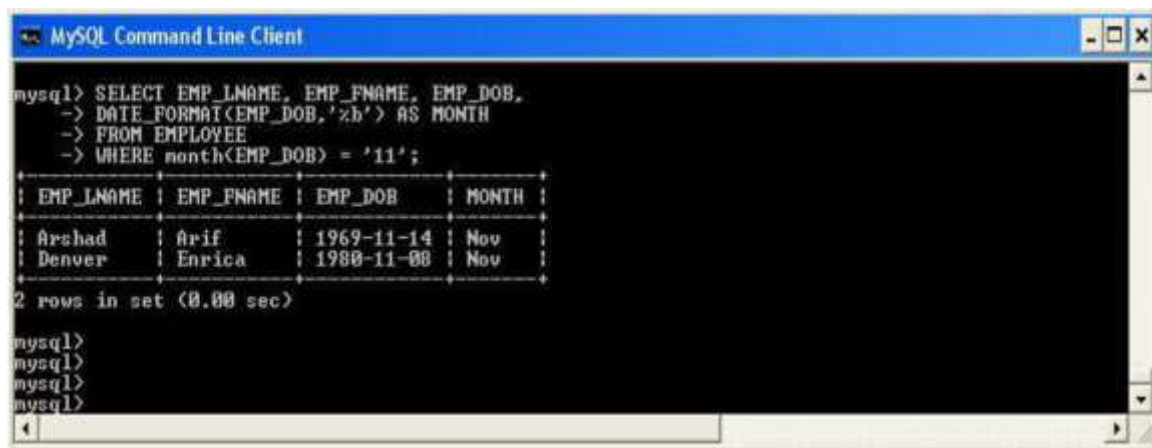
7 rows in set (0.00 sec)

mysql>

```

Figure 57 Using the MONTH, DAYOFMONTH and YEAR functions.

Task 6.4 Write a query that displays all employees who were born in November. Your output should match that shown in Figure 58.



```

mysql> SELECT EMP_LNAME, EMP_FNAME, EMP_DOB,
-> DATE_FORMAT(EMP_DOB, '%b') AS MONTH
-> FROM EMPLOYEE
-> WHERE month(EMP_DOB) = '11';

```

EMP_LNAME	EMP_FNAME	EMP_DOB	MONTH
Arshad	Arif	1969-11-14	Nov
Denver	Enrica	1980-11-08	Nov

```

2 rows in set (0.00 sec)

mysql>
mysql>
mysql>
mysql>

```

Figure 58 Output for Task 6.4.

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 6.5 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 ADDBDATE('2008-01-01', INTERVAL 11 MONTH );
```

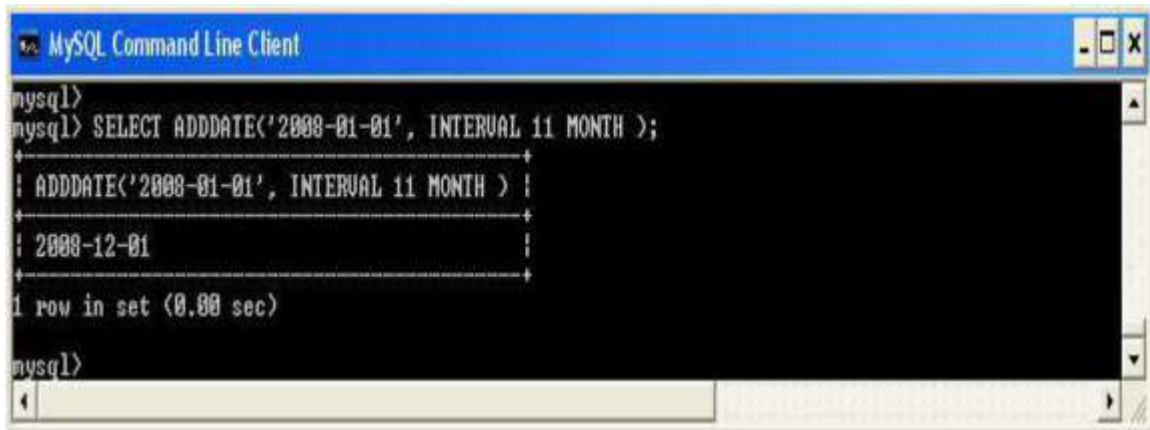


Figure 59 Adding months to a date

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

Task 6.6 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,  
ADDBDATE(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 6.7 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;
```

6.4 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

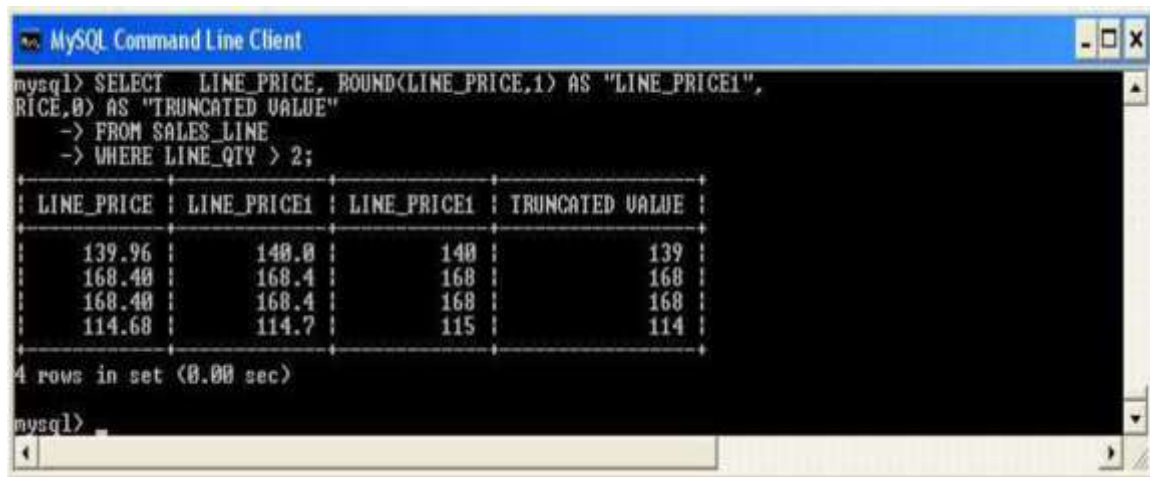
Function	Description
ABS	Returns the absolute value of a number Syntax: ABS(numeric_value)
ROUND	Rounds a value to a specified precision (number of digits)

	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.



LINE_PRICE	LINE_PRICE1	LINE_PRICE1	TRUNCATED VALUE
139.96	140.0	140	139
168.40	168.4	168	168
168.40	168.4	168	168
114.68	114.7	115	114

4 rows in set (0.00 sec)

Figure 60 Example of ROUND and TRUNC

Task 6.8 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;
```

6.5 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.

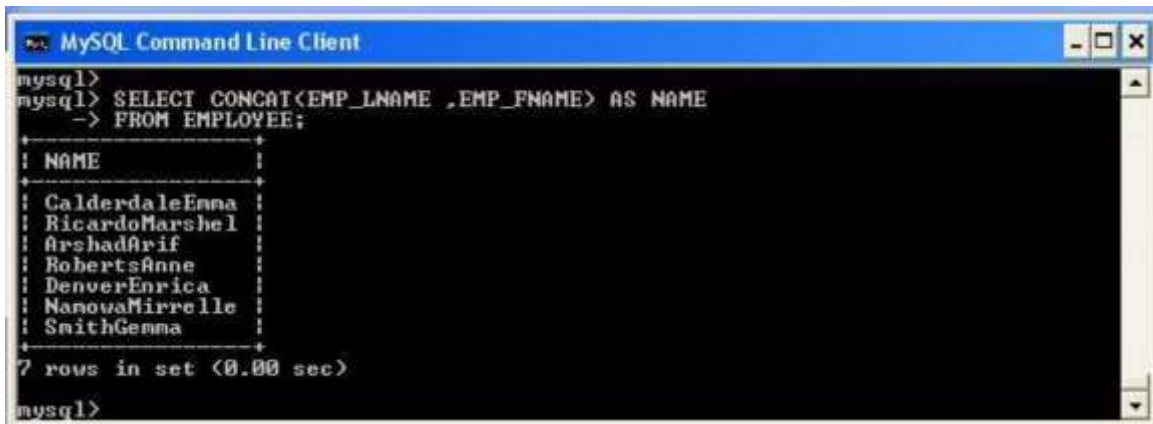
Function	Description
CONCAT	Concatenates data from two different character columns and returns a single column. Syntax: CONCAT(strg_value, strg_value)
UPPER/LOWER	Returns a string in all capital or all lowercase letters Syntax: UPPER(strg_value) , LOWER(strg_value)
SUBSTR	Returns a substring or part 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;
```



The screenshot shows a MySQL Command Line Client window. The prompt is 'mysql>'. The user enters the query: 'SELECT CONCAT(EMP_LNAME ,EMP_FNAME) AS NAME FROM EMPLOYEE;'. The output is a table with one column named 'NAME' and seven rows of concatenated names. The output is displayed in a table format with dashed lines around the data.

NAME
CalderdaleEmma
RicardoMarshall
ArshadArif
RobertsAnne
DenverEnrica
NamouaMirrelle
SmithGemma

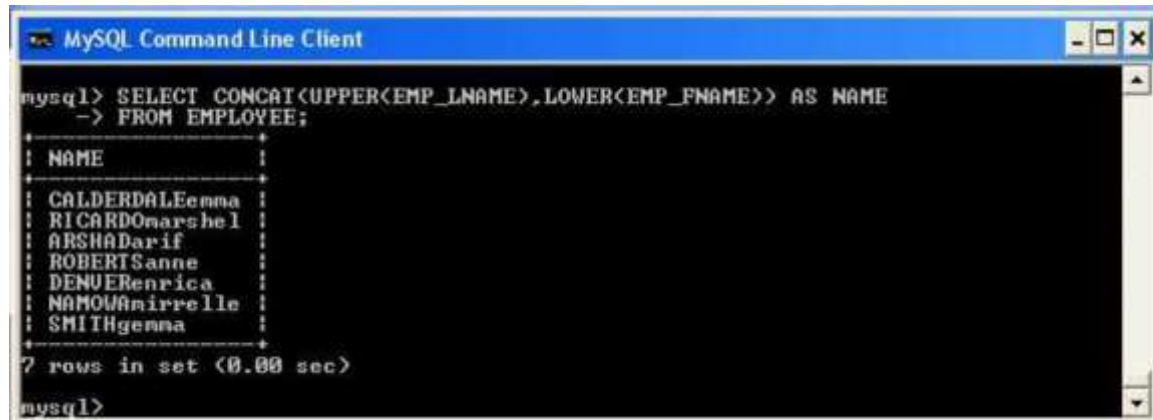
7 rows in set (0.00 sec)

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;
```



The screenshot shows a MySQL Command Line Client window. The prompt is 'mysql>'. The user enters the query: 'SELECT CONCAT(UPPER(EMP_LNAME),LOWER(EMP_FNAME)) AS NAME FROM EMPLOYEE;'. The output is a table with one column named 'NAME' and seven rows of concatenated names where the last name is in uppercase and the first name is in lowercase. The output is displayed in a table format with dashed lines around the data.

NAME
CALDERDALEemma
RICARDOMarshall
ARSHADarif
ROBERTSanne
DENVERenrica
NAMOUAmirrelle
SMITHgemma

7 rows in set (0.00 sec)

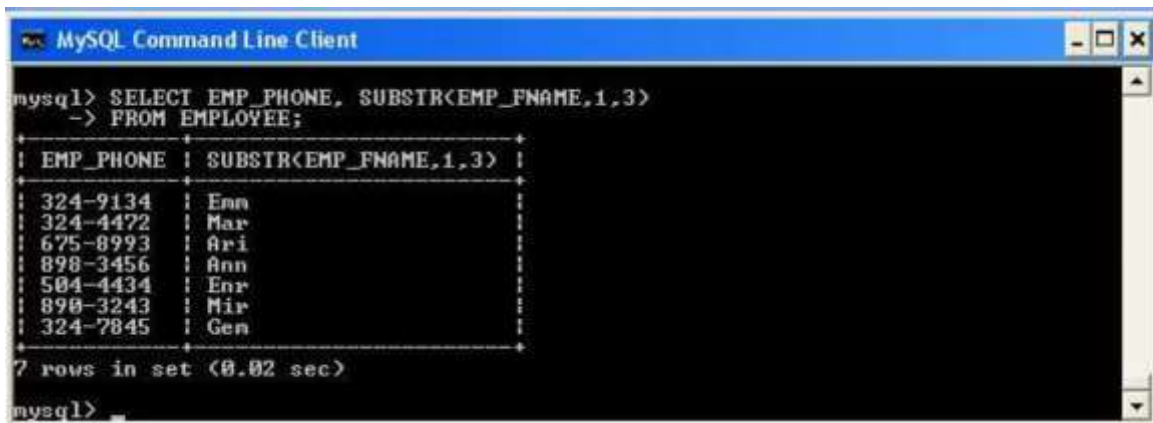
Figure 62 Displaying upper and lower case employee names.

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;
```



The screenshot shows a MySQL Command Line Client window. The command entered is `mysql> SELECT EMP_PHONE, SUBSTR(EMP_FNAME,1,3) -> FROM EMPLOYEE;`. The output is a table with two columns: `EMP_PHONE` and `SUBSTR(EMP_FNAME,1,3)`. The data rows are:

EMP_PHONE	SUBSTR(EMP_FNAME,1,3)
324-9134	Enn
324-4472	Mar
675-8993	Ari
898-3456	Ann
504-4434	Enr
890-3243	Mir
324-7845	Gen

7 rows in set (0.02 sec)

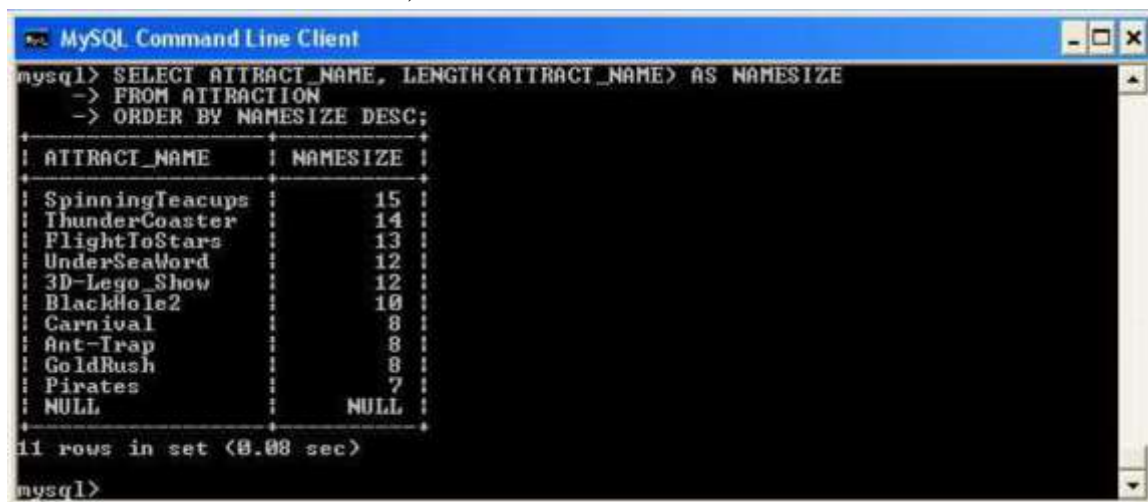
Figure 63 Displaying the first 3 characters of the employees first name

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;
```



The screenshot shows a MySQL Command Line Client window. The command entered is `mysql> SELECT ATTRACT_NAME, LENGTH(ATTRACT_NAME) AS NAMESIZE -> FROM ATTRACTION -> ORDER BY NAMESIZE DESC;`. The output is a table with two columns: `ATTRACT_NAME` and `NAMESIZE`. The data rows are:

ATTRACT_NAME	NAMESIZE
SpinningTeacups	15
ThunderCoaster	14
FlightToStars	13
UnderSeaWord	12
3D-Lego_Show	12
BlackHole2	10
Carnival	8
Ant-Trap	8
GoldRush	8
Pirates	7
NULL	NULL

11 rows in set (0.00 sec)

Figure 65 Displaying the length of attraction names.

6.5 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:

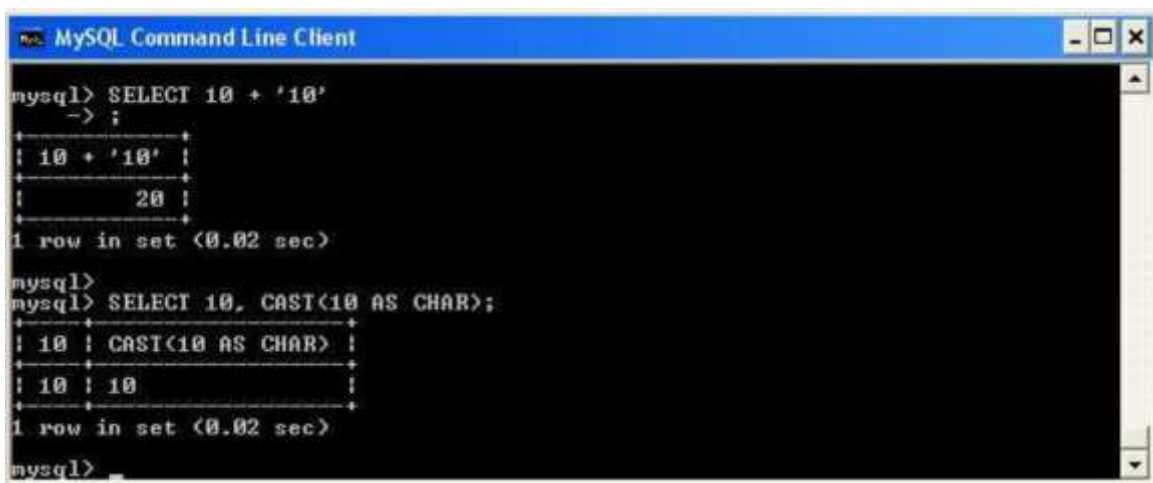
```
SELECT 10 + '10'
```

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 CONVERT 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);
```



```
mysql> SELECT 10 + '10'
-> ;
+-----+
| 10 + '10' |
+-----+
|         20 |
+-----+
1 row in set (0.02 sec)

mysql>
mysql> SELECT 10, CAST(10 AS CHAR);
+-----+-----+
| 10 | CAST(10 AS CHAR) |
+-----+-----+
| 10 | 10                |
+-----+-----+
1 row in set (0.02 sec)

mysql>
```

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 conversion 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.

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.

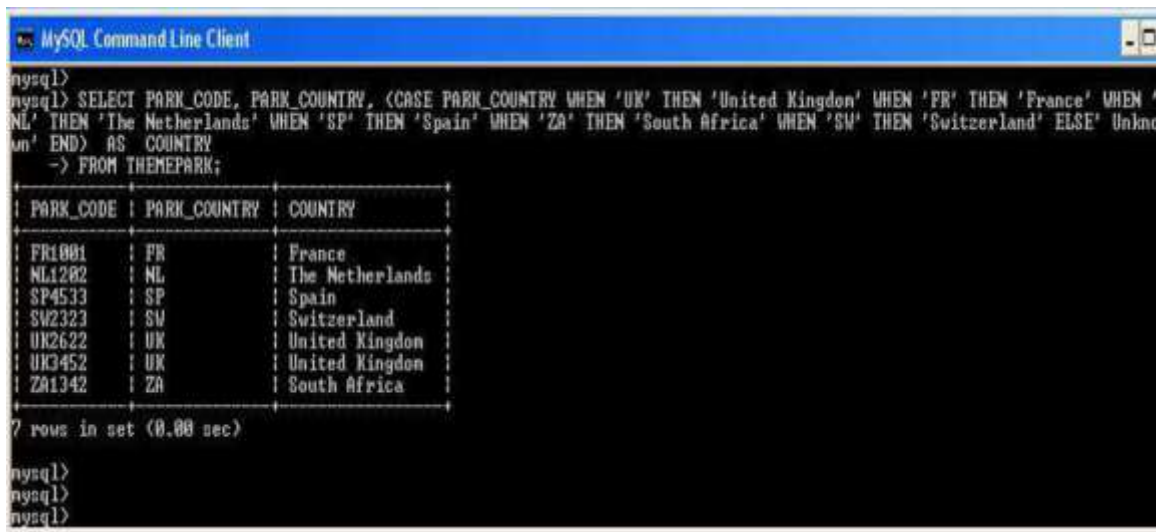
CASE value WHEN [compare_value] THEN result [WHEN [compare_value] THEN result ...] [ELSE result] END

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;
```



```
mysql> 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;
```

PARK_CODE	PARK_COUNTRY	COUNTRY
FR1001	FR	France
NL1202	NL	The Netherlands
SP4533	SP	Spain
SW2323	SW	Switzerland
UK2622	UK	United Kingdom
UK3452	UK	United Kingdom
ZA1342	ZA	South Africa

```
7 rows in set (0.00 sec)

mysql>
mysql>
mysql>
```

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;
```

Exercises

E6.1 Create the view EMPFR (as created in section 6.2) and update the Theme Park that employee number 101 works in. (Update the employee number 101 information in the EMPFR view).

E6.2 Employee Emma Cauderdale (EMP_NUM =100) has now changed her phone number to 324-9652. Update her information in the EMPFR view. Write a query to show her new phone number has been updated and then Remove the EMPFR view.

E6.3 Create a view of only those Theme Parks where tickets have been sold and then display the contents of this view. Your output should match that shown in Figure E-6.4.

```
+-----+-----+-----+-----+
| PARK_CODE | PARK_NAME | PARK_CITY | PARK_COUNTRY |
+-----+-----+-----+-----+
| FR1001    | FairyLand | PARIS     | FR           |
| UK3452    | PleasureLand | STOKE    | UK           |
| ZA1342    | GoldTown  | JOHANNESBURG | ZA          |
+-----+-----+-----+-----+
3 rows in set (0.08 sec)

mysql> describe tparkssold;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| PARK_CODE  | varchar(10) | NO   |     | NULL    |       |
| PARK_NAME  | varchar(35) | NO   |     | NULL    |       |
| PARK_CITY  | varchar(50) | NO   |     | NULL    |       |
| PARK_COUNTRY | char(2)    | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.25 sec)
```

Figure 1 Output for E-6.3

E6.4 The Theme Park managers want to create a view called EMP_DETAILS which contains the following information. EMP_NO, PARK_CODE, PARK_NAME, EMP_LNAME_EMP_FNAME, EMP_HIRE_DATE and EMP_DOB. The view should only be read only. Check that the view works, by displaying its contents.

```
mysql> describe emp_details;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| emp_num | decimal(4,0) | NO | | NULL | |
| park_code | varchar(10) | YES | | NULL | |
| park_name | varchar(35) | NO | | NULL | |
| emp_lname | varchar(15) | NO | | NULL | |
| emp_fname | varchar(15) | NO | | NULL | |
| emp_hire_date | date | YES | | NULL | |
| emp_dob | date | NO | | NULL | |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql> select * from emp_details;
+-----+-----+-----+-----+-----+-----+-----+
| emp_num | park_code | park_name | emp_lname | emp_fname | emp_hire_date | emp_dob |
+-----+-----+-----+-----+-----+-----+-----+
| 100 | FR1001 | FairyLand | Calderdale | Emma | 1992-03-15 | 1972-06-15 |
| 101 | UK3452 | PleasureLand | Ricardo | Marshal | 1996-04-25 | 1978-03-19 |
| 102 | FR1001 | FairyLand | Arshad | Arif | 1990-12-20 | 1969-11-14 |
| 103 | UK3452 | PleasureLand | Roberts | Anne | 1994-08-16 | 1974-10-16 |
| 104 | ZA1342 | GoldTown | Denver | Enrica | 2001-10-20 | 1980-11-08 |
| 105 | FR1001 | FairyLand | Namowa | Mirrelle | 2006-11-08 | 1990-03-14 |
| 106 | ZA1342 | GoldTown | Smith | Gemma | 1989-01-05 | 1968-02-12 |
+-----+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

Figure 2 Output for E-6.4

E6.5 Using your view EMP_DETAILS, write a query that displays all employee first and last names and the park names.

E6.6 Create a view called TICKET_SALES which contains details of the min, max and average sales at each Theme Park. The name of the theme park should also be displayed.

(Hint 1: you will need to join three tables). Once you have created your view, write a query to display the contents.

```
+-----+-----+-----+
| park_name | min(line_price) | max(line_price) |
+-----+-----+-----+
| FairyLand | 14.99 | 139.96 |
| GoldTown | 12.12 | 114.68 |
| PleasureLand | 21.98 | 168.40 |
+-----+-----+-----+
3 rows in set (0.01 sec)
```

Figure 3 Output for E6.6

E6.7 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'.

E 6.8 Write a query which generates a list of employee user IDs, using the born month, 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.

USER ID format (**MDName**) here M= month, D= first day of month, Name= Employee last name first 6 alphabets.

emp_fname	emp_lname	user_id
Emma	Calderdale	0601CALDER
Marshall	Ricardo	0301RICARD
Arif	Arshad	1101ARSHAD
Anne	Roberts	1001ROBERT
Enrica	Denver	1101DENVER
Mirrelle	Namowa	0301NAMOWA
Gemma	Smith	0201SMITH

7 rows in set (0.00 sec)

Figure 64 Results for E 6.9.

E6.9 Write a query which lists the names and dates of births of all employees born on the 14th day of the month.

E6.10 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;

References:

<https://www.mysqltutorial.org/mysql-views-tutorial.aspx/>
<https://dev.mysql.com/doc/refman/8.0/en/sql-function-reference.html>