Lab 3: Basic SELECT statements

The learning objectives of this lab are to

- Use arithmetic operators in SQL statements
- Select rows from a table with conditional restrictions
- Apply logical operators to have multiple conditions
- Sort the data in the resulting query

3.1 Using arithmetic operators in SQL statements

SQL commands are often used in conjunction with arithmetic operators. As you perform mathematical operations on attributes, remember the rules of precedence. As the name suggests, the **rules of precedence** are the rules that establish the order in which computations are completed. For example, note the order of the following computational sequence:

1. Perform operations within parentheses
2. Perform power operations
3. Perform multiplications and divisions
4. Perform additions and subtractions

Task 3.1 Suppose the owners of all the theme parks wanted to compare the current ticket prices, with an increase in the price of each ticket by 10%. To generate this query type:

```
SELECT PARK_CODE, TICKET_NO, TICKET_TYPE, TICKET_PRICE,
TICKET_PRICE + ROUND((TICKET_PRICE *0.1),2)
```

FROM TICKET;

The output for this query is shown in Figure 19. The ROUND function is used to ensure the result is displayed to two decimal places.

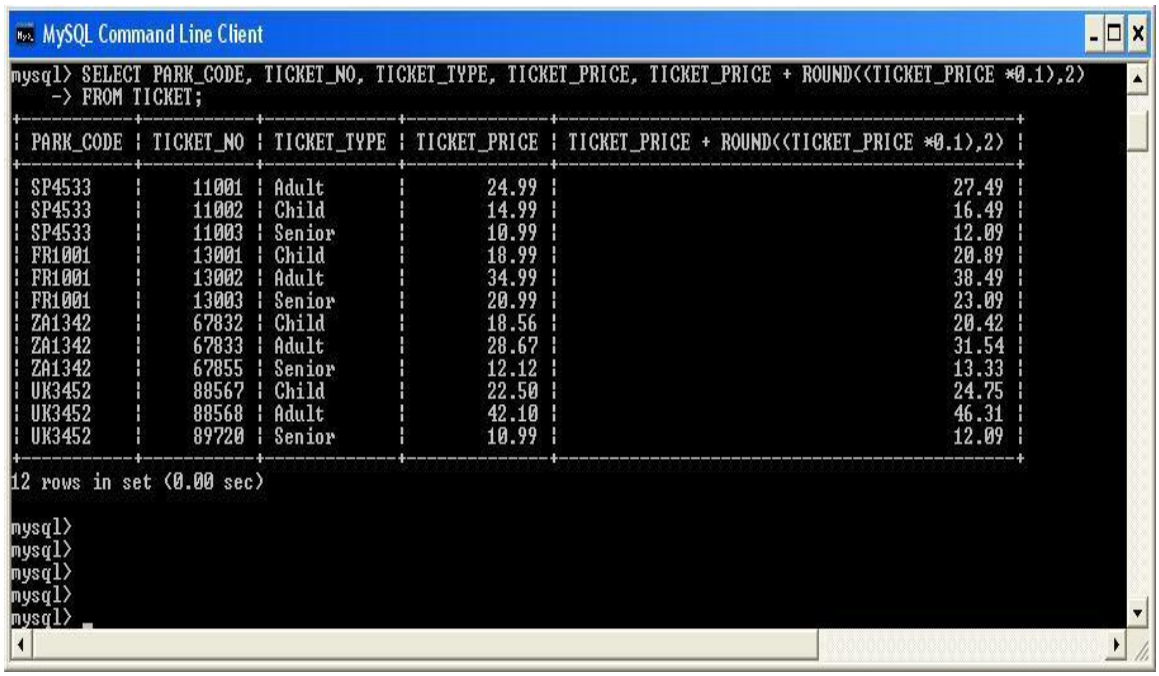


Figure 19: Output showing 10% increase in ticket prices

You will see in Figure 19 that the last column is named after the arithmetic expression in the query. To rename the column heading, a column alias needs to be used. Modify the query as follows and note that the name of the heading has changed to PRICE_INCREASE when you execute the following query.

```
SELECT PARK_CODE, TICKET_NO, TICKET_TYPE, TICKET_PRICE,
TICKET_PRICE + ROUND((TICKET_PRICE *0.1),2) PRICE_INCREASE
FROM TICKET;
```

Note

When dealing with column names that require spaces, the optional keyword AS can be used. For example:

```
SELECT PARK_CODE, TICKET_NO, TICKET_TYPE,
TICKET_PRICE, TICKET_PRICE + ROUND((TICKET_PRICE *0.1),2)
AS “PRICE INCREASE”
FROM TICKET;
```

3.2 Selecting rows with conditional restrictions

Numerous conditional restrictions can be placed on the selected table contents in the WHERE clause of the SELECT statement. For example, the comparison operators shown in Table 1 can be used to restrict output.

Table 1 Comparison Operators

SYMBOL	**MEANING**
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to
BETWEEN	Used to check if an attribute is within a range.
IN	Used to check if an attribute value matches any value within a list.

LIKE	Used to check if an attribute value matches a given string pattern.
IS NULL / IS NOT NULL	Used to check if an attribute is NULL / is not NULL.

We will now explore some of these conditional operators using examples.

Greater than

The following example uses the "greater than" operator to display the theme park code, ticket price and ticket type of all tickets where the ticket price is greater than €20.00.

```
SELECT PARK_CODE, TICKET_TYPE, TICKET_PRICE
FROM TICKET
WHERE TICKET_PRICE > 20;
```

The output is shown in Figure 20.

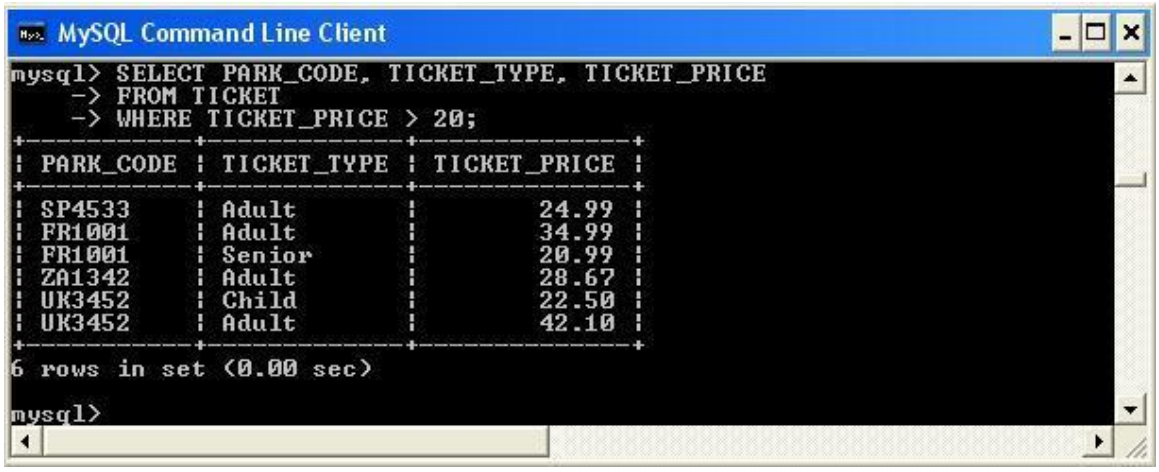


Figure 20: Tickets costing greater than €20.00

Task 3.2 Type in and execute the query and test out the greater than operator. Do you get the same results has shown in Figure 20?

Task 3.3 Modify the query you have just executed to display tickets that are less than €30.00.

Character comparisons

Comparison operators may even be used to place restrictions on character-based attributes.

Task 3.4 Execute the following query which produces a list of all rows in which the PARK_CODE is alphabetically less than UK2262. (Because the ASCII code value for the letter *B* is greater than the value of the letter *A*, it follows that *A* is less than *B*.) Therefore, the output will be generated as shown in Figure 21.

```
SELECT      PARK_CODE, PARK_NAME, PARK_COUNTRY
FROM        THEMEPARK
WHERE       PARK_CODE < 'UK2262';
```

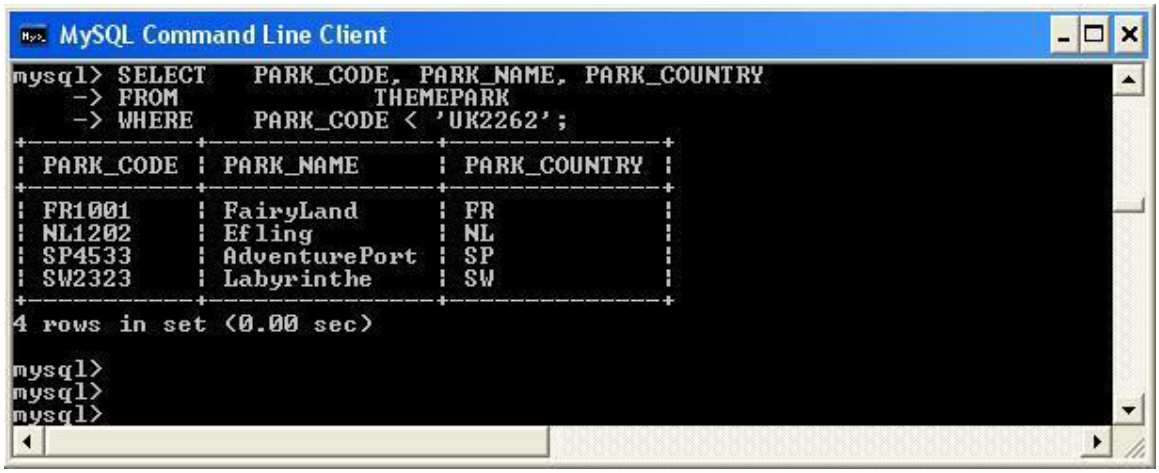


Figure 21: Example of character comparison

BETWEEN

The operator BETWEEN may be used to check whether an attribute value is within a range of values. For example, if you want to see a listing for all tickets whose prices are between €30 and €50, use the following command sequence:

```
SELECT      *
FROM        TICKET
WHERE       TICKET_PRICE BETWEEN 30.00 AND 50.00;
```

Figure 22 shows the output you should see for this query.

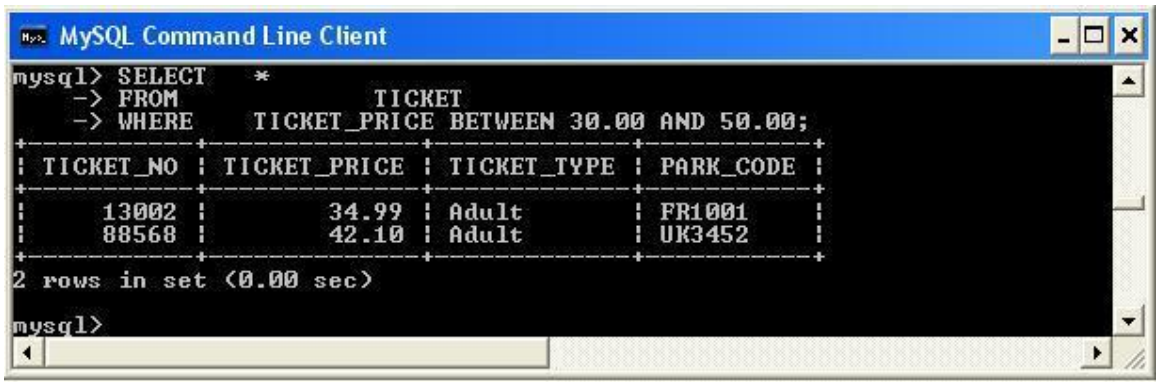


Figure 22: Displaying ticket prices BETWEEN two values.

Task 3.5 Write a query which displays the employee number, attraction no, the hours worked per attraction and the date worked where the hours worked per attraction is between 5 and 10. Hint you will need to select data from the HOURS table. The output for the query is shown in Figure 23.

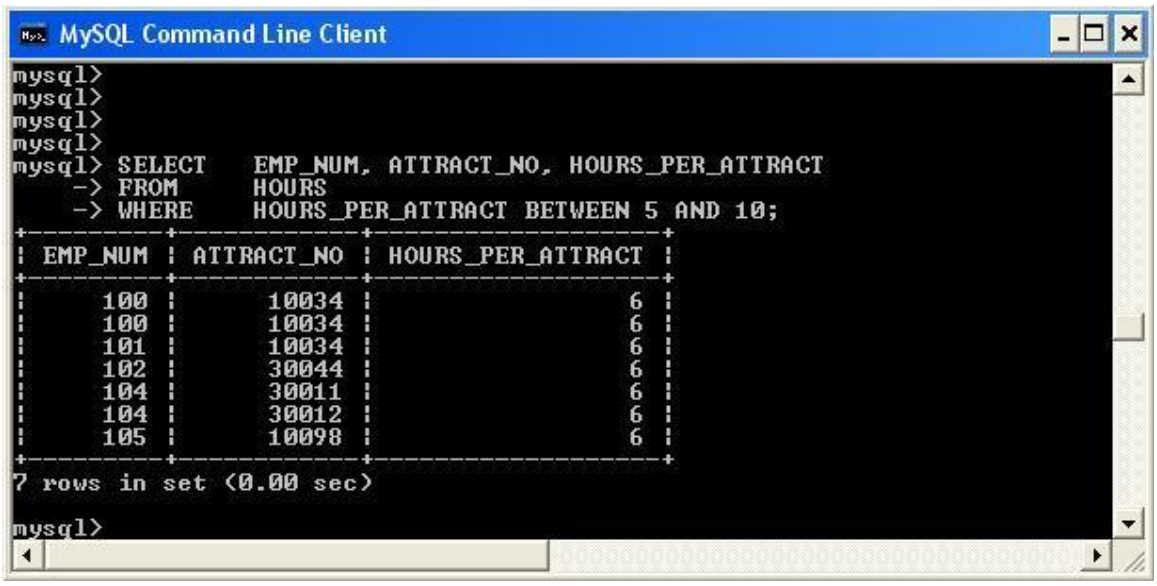


Figure 23: Output for Task 3.5

IN

The IN operator is used to test for values which are in a list. The following query finds only the rows in the SALES_LINE table that match up to a specific sales transaction. i.e. TRANSACTION_NO is either 12781 or 67593.

```
SELECT      *
FROM        SALES_LINE
WHERE       TRANSACTION_NO IN (12781, 67593);
```

The result of this query is shown in Figure 24.

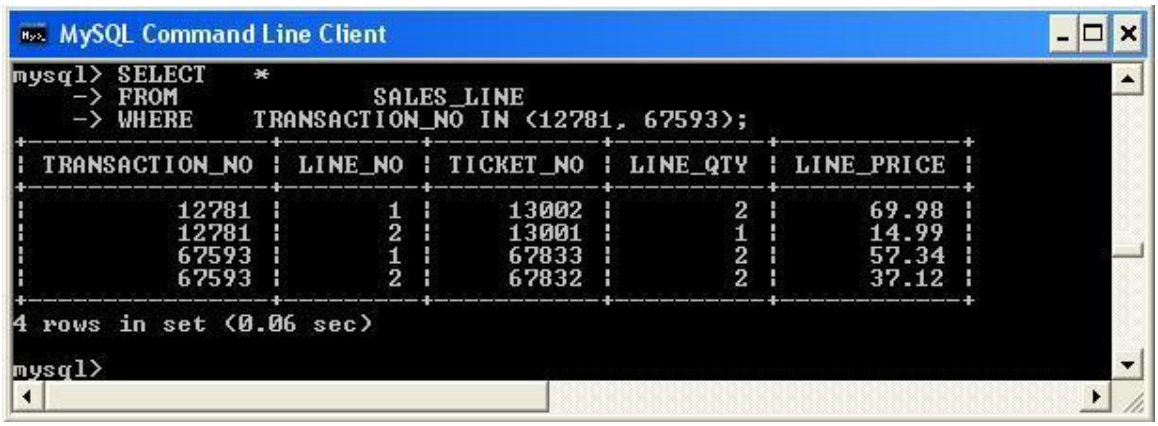


Figure 24 Selecting rows using the IN command

Task 3.6 Write a query to display all tickets that are of type Senior or Child. Hint:

Use the TICKET table. The output you should see is shown in Figure 25.

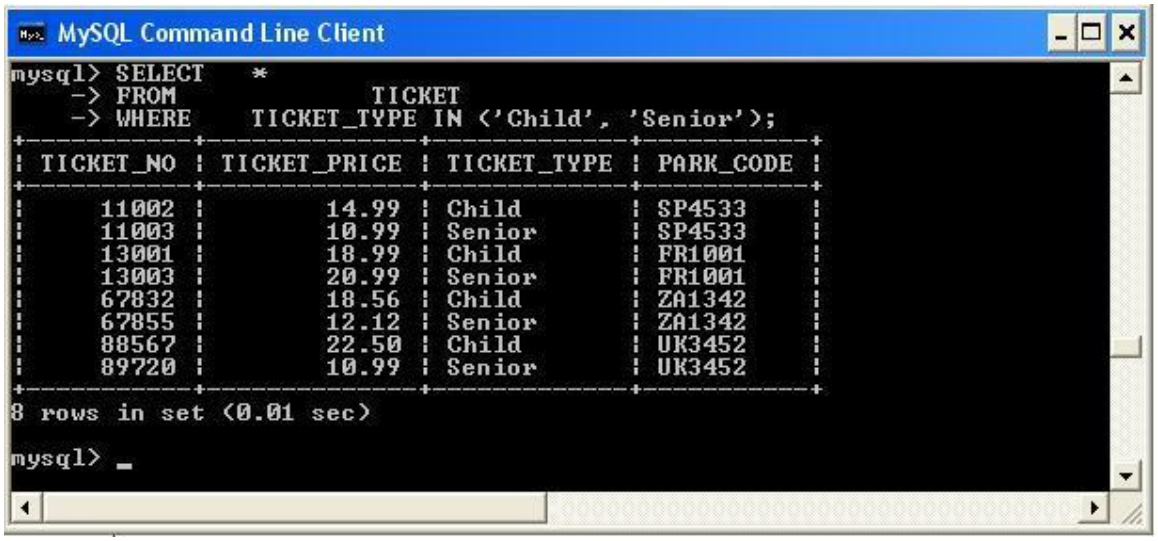


Figure 25. Output for Task 3.6

LIKE

The LIKE operator is used to find patterns within string attributes. Standard SQL allows you to use the percent sign (%) and underscore (_) wildcard characters to make matches

when the entire string is not known. % means any and all *following* characters are eligible while _ means any *one* character may be substituted for the underscore.

Task 3.7 Enter the following query which finds all EMPLOYEE rows whose first names begin with the letter *A*.

```
SELECT    EMP_LNAME, EMP_FNAME, EMP_NUM
FROM      EMPLOYEE
WHERE     EMP_FNAME LIKE 'A%';
```

Figure 26 shows the output you should see for this query.

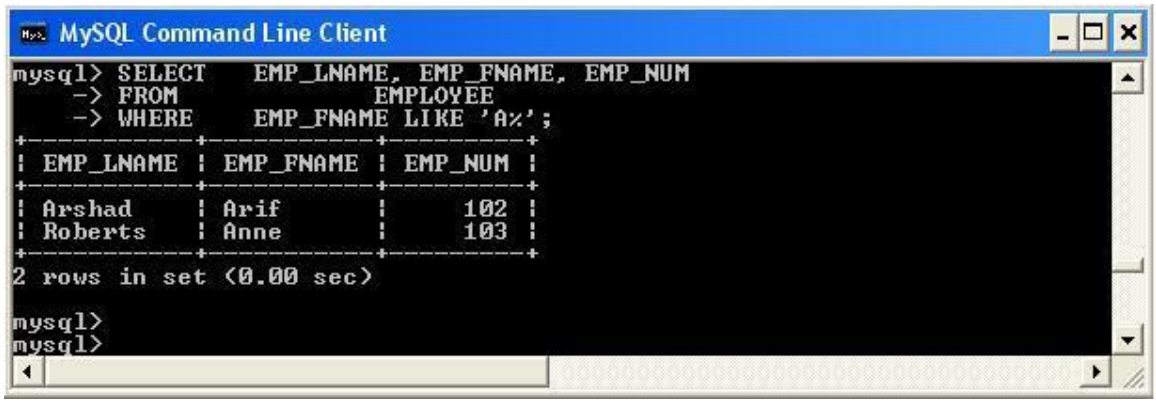


Figure 26 Query using the LIKE command

Task 3.8 Write a query which finds all Theme Parks that have a name ending in 'Land'. The output you should see is shown in Figure 27.

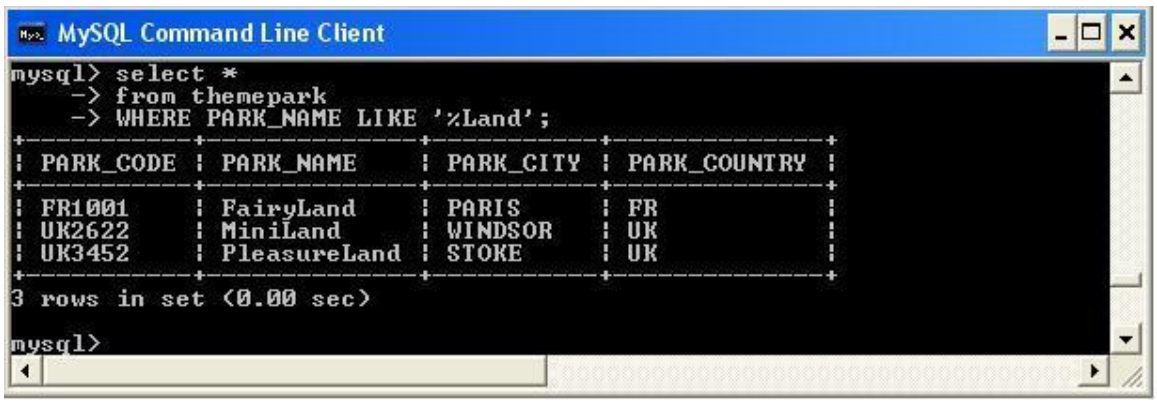


Figure 27 Solution to Task 4.8

NULL and IS NULL

IS NULL is used to check for a null attribute value. In the following example, the query lists all attractions that do not have an attraction name assigned (ATTRACT_NAME is null). The query could be written as:

SELECT *

FROM ATTRACTION

WHERE ATTRACT_NAME IS NULL;

The output for this query is shown in Figure 28.

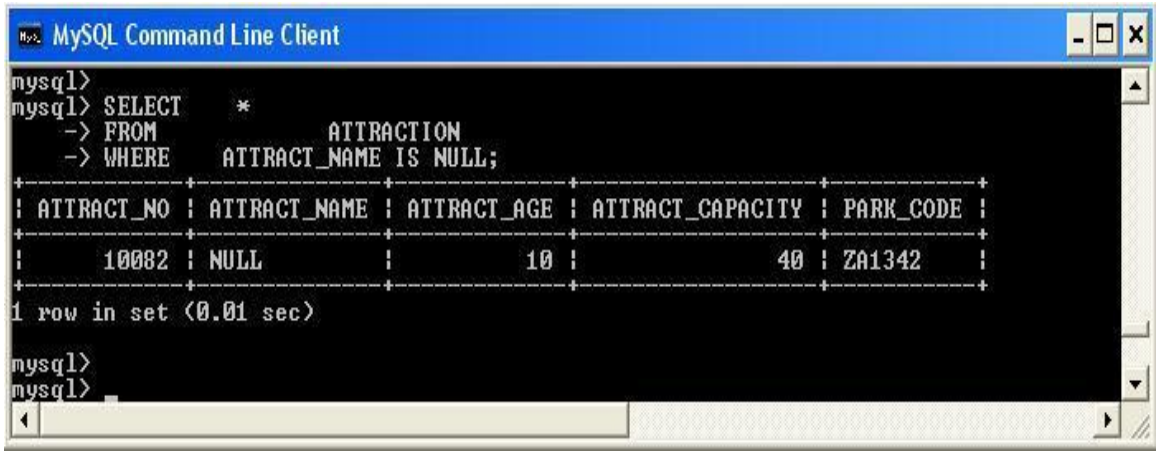


Figure 28 Listing all Attractions with no name

Logical Operators

SQL allows you to have multiple conditions in a query through the use of logical operators: AND, OR and NOT. NOT has the highest precedence, followed by AND, and then followed by OR. However, you are strongly recommended to use parentheses to clarify the intended meaning of the query.

AND

This logical AND connective is used to set up a query where there are two conditions which must be met for the query to return the required row(s). The following query displays the employee number (EMP_NUM) and the attraction number (ATTRACT_NUM) for which the numbers of hours worked (HOURS_PER_ATTRACT) by the employee is greater than 3 and the date worked (DATE_WORKED) is after 18th May 2007.

```
SELECT      EMP_NUM, ATTRACT_NO
FROM        HOURS
WHERE       HOURS_PER_ATTRACT > 3
AND         DATE_WORKED > '2007-05-18';
```

This query will produce the output shown in Figure 29.

```
mysql>
mysql>
mysql>
mysql> SELECT EMP_NUM, ATTRACT_NO
    -> FROM HOURS
    -> WHERE HOURS_PER_ATTRACT > 3
    -> AND DATE_WORKED > '2007-05-18';
+---------+------------+
| EMP_NUM | ATTRACT_NO |
+---------+------------+
|     100 |      10034 |
|     102 |      30044 |
|     104 |      30011 |
|     104 |      30012 |
|     105 |      10098 |
+---------+------------+
5 rows in set (0.00 sec)

mysql>
```

Figure 29 Query results using the AND operator

Task 3.9 Enter the query above and check you results with those shown in Figure 29.

Task 3.10 Write a query which displays the details of all attractions which are suitable for children aged 10 or under and have a capacity of less than 100. You should not display any information for attractions which currently have no name. Your output should correspond to that shown in Figure 30.

```
MySQL Command Line Client
mysql> SELECT   *
    -> FROM     ATTRACTION
    -> WHERE    ATTRACT_AGE <= 10
    -> AND      ATTRACT_CAPACITY < 100
    -> AND ATTRACT_NAME IS NOT NULL;
+------------+----------------+-------------+-----------------+-----------+
| ATTRACT_NO | ATTRACT_NAME   | ATTRACT_AGE | ATTRACT_CAPACITY | PARK_CODE |
+------------+----------------+-------------+-----------------+-----------+
|      10056 | SpinningTeacups |           4 |              62 | FR1001    |
|      30012 | Pirates        |          10 |              42 | UK3452    |
|      30044 | UnderSeaWord   |           4 |              80 | UK3452    |
|      98764 | GoldRush       |           5 |              80 | ZA1342    |
+------------+----------------+-------------+-----------------+-----------+
4 rows in set (0.00 sec)

mysql>
```

Figure 30: Query results for Task 3.10

OR

If you wanted to list the names and countries of all Theme parks where of invoice numbers where PARK_COUNTRY = 'FR' OR PARK_COUNTRY = 'UK' you would write the following query.

```
SELECT PARK_NAME, PARK_COUNTRY
FROM THEMEPARK
WHERE PARK_COUNTRY = 'FR'
OR PARK_COUNTRY = 'UK';
```

The output is shown in Figure 31.

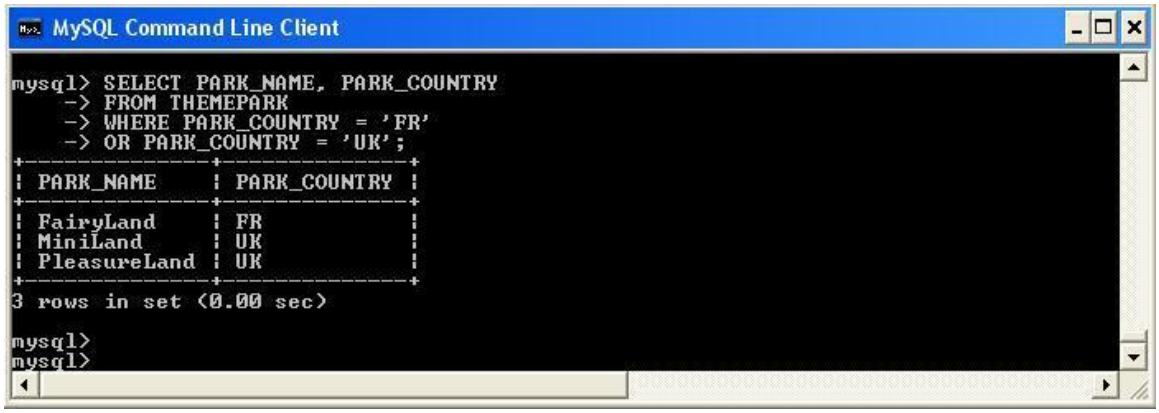


Figure 31: Query results using the OR operator;

When using AND and OR in the same query it is advisable to use parentheses to make explicit the precedence.

Task 3.11 Test the following query and check your output with that shown in Figure 32. Can you work out what this query is doing?

```
SELECT      *
FROM        ATTRACTION
WHERE       (PARK_CODE LIKE 'FR%'
AND ATTRACT_CAPACITY <50) OR (ATTRACT_CAPACITY > 100);
```

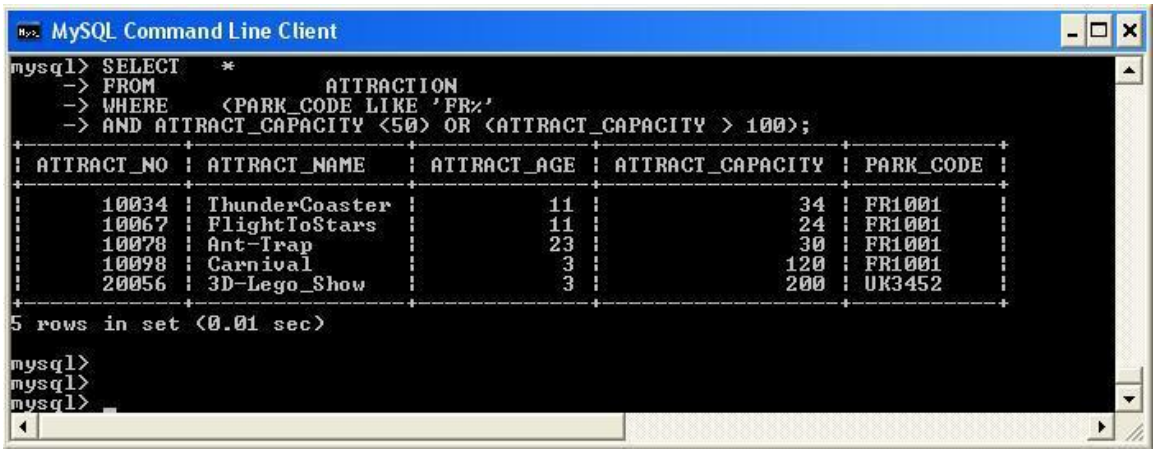


Figure 32: AND and OR example

NOT

The logical operator **NOT** is used to negate the result of a conditional expression. If you want to see a listing of all rows for which EMP_NUM is not 106, the query would look like:

```
SELECT      *
FROM EMPLOYEE
WHERE       NOT (EMP_NUM = 106);
```

The results of this query are shown in Figure 33. Note that the condition is enclosed in parentheses; that practice is optional, but it is highly recommended for clarity.

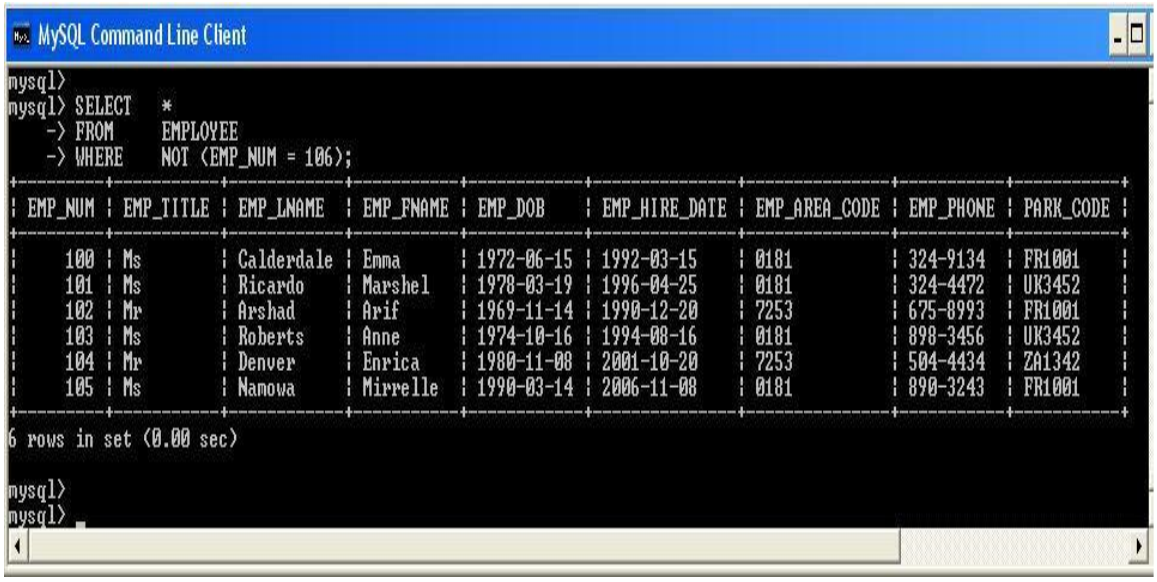


Figure 33: Listing all employees except EMP_NUM=106

3.3 Sorting Data

The **ORDER BY** clause is especially useful when the listing order of the query is important. Although you have the option of declaring the order type—ascending (**ASC**) or descending (**DESC**) —the default order is ascending. For example, if you want to display all employees listed by EMP_HIRE_DATE in descending order you would write the following query. The output is shown in Figure 34.

```
SELECT      *
FROM        EMPLOYEE
ORDER BY    EMP_HIRE_DATE DESC;
```

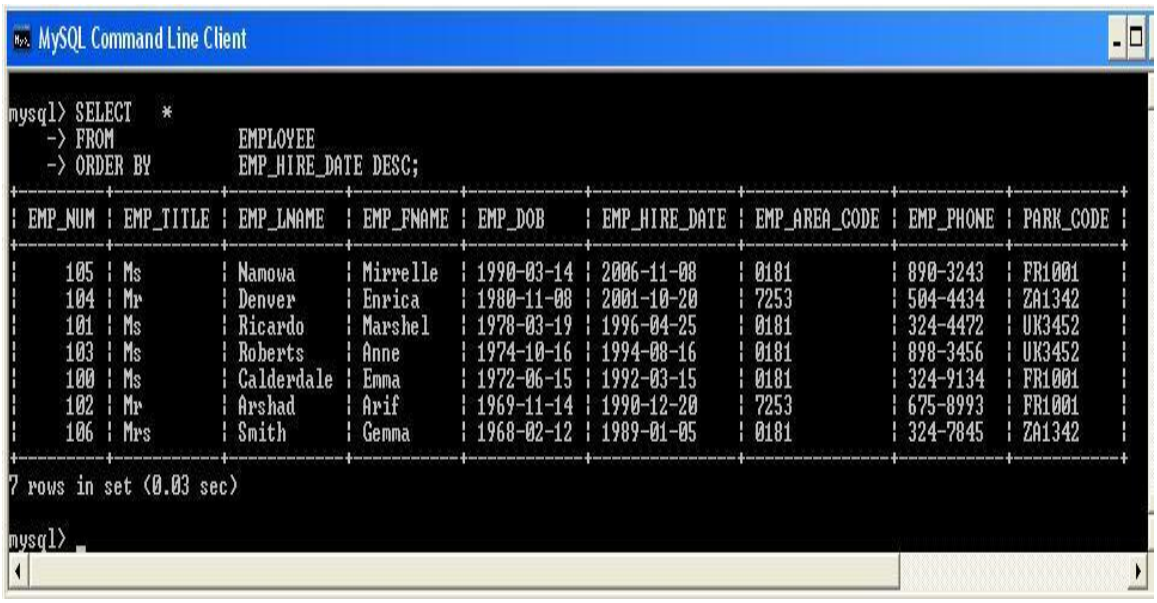


Figure 34: Displaying all employees in descending order of EMP_HIRE_DATE.

The ORDER BY command can also be used to produce a cascading order sequence. This is where the query results are ordered against a sequence of attributes.

Task 3.12 Enter the following query which contains an example of a cascading order sequence, by ordering the rows in the employee table by the employee's last then first names.

```
SELECT      *
FROM        EMPLOYEE
ORDER BY    EMP_LNAME, EMP_FNAME;
```

It is worth noting that if the ordering column has nulls, they are listed either first or last (depending on the RDBMS). The ORDER BY clause can be used in conjunction with other SQL commands and is listed last in the SELECT command sequence.

Task 3.13 Enter the following query and check your output against the results shown in Figure 35. Describe in your own words what this query is actually doing.

```
SELECT        TICKET_TYPE, PARK_CODE
FROM TICKET
WHERE (TICKET_PRICE > 15 AND TICKET_TYPE LIKE 'Child')
ORDER BY TICKET_NO DESC;
```

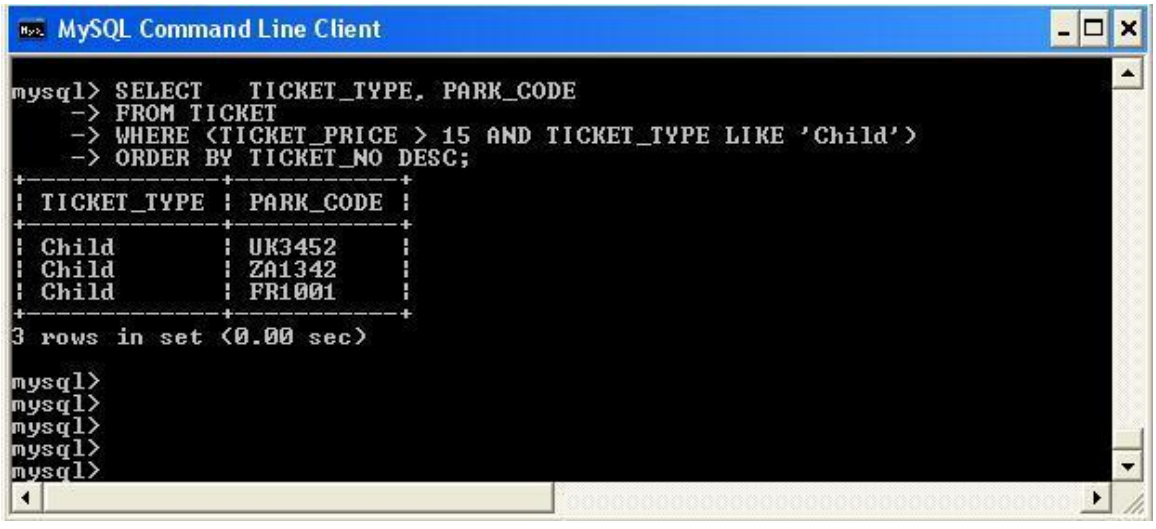


Figure 35: Query results for Task 5.2.

3.4 Listing Unique Values

The SQL command DISTINCT is used to produce a list of only those values that are different from one another. For example to list only the different Theme parks from within the ATTRACTION table, you would enter the following query.

```
SELECT      DISTINCT(PARK_CODE)

FROM        ATTRACTION;
```

Figure 36 shows that the query only displays the rows that are different.

Figure 36: Displaying DISTINCT rows.

Exercises

E3.1 Write a query to display all Theme Parks except those in the UK.

E3.2 Write a query to display all the sales that occurred on the 18th May 2007.

E3.3 Write a query to display the ticket prices between €20 AND €30.

E3.4 Display all attractions that have a capacity of more than 60 at the Theme Park

FR1001.

E3.5 Write a query to display the hourly rate for each attraction where an employee had

worked, along with the hourly rate increased by 20%. Your query should only

Display the ATTRACT_NO, HOUR_RATE and the HOUR_RATE with the 20%

Increase.

E.3.6 Elaborate Difference (IN vs BETWEEN) operators with examples.

E.3.7 Write a query to display all unique employees that exist in the HOURS table.

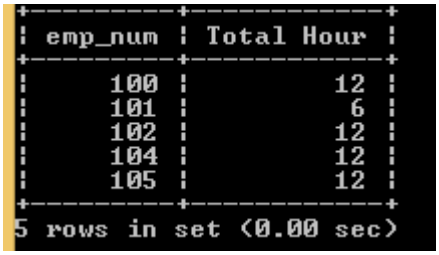
E.3.8 Display all information from the SALES table in descending order of the sale date.

E.3.9 Write a query to show the transaction numbers and lineprices (in the SALES_LINE table) that are greater than €50.

E.3.10 Write a query to display only the last two Employee Record (EMP_NUM,EMP_FNAME) in descending order.

Bonus Task

1. Display the employee numbers of all employees and the total number of hours they have worked Check your result with those shown in below figure 37.



emp_num	Total Hour
100	12
101	6
102	12
104	12
105	12

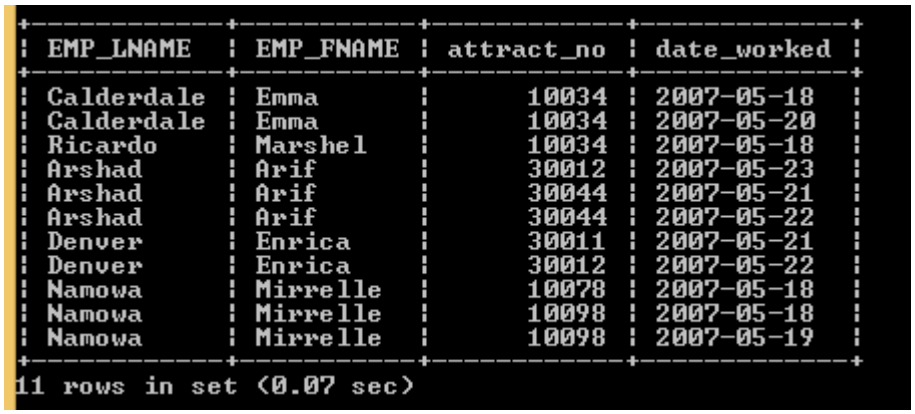
5 rows in set (0.00 sec)

Figure 37. Output Format

2. Write a query that displays the employees first and last name (EMP_FNAME and EMP_LNAME), the attraction number(ATTRACT_NO) and the date worked.

Hint:

You will have to join the HOURS and the EMPLOYEE tables. Check your results with those shown in below figure 38.



EMP_LNAME	EMP_FNAME	attract_no	date_worked
Calderdale	Emma	10034	2007-05-18
Calderdale	Emma	10034	2007-05-20
Ricardo	Marshel	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 (0.07 sec)

Figure 38. Output Format