

Lab 8: Subqueries

The learning objectives of this lab are to

- Learn how to use subqueries to extract rows from processed data
- Select the most suitable subquery format
- Use correlated subqueries

First let's outline the basic characteristics of a subquery, which were introduced in Chapter 8, Introduction to Structured Query Language.

- A subquery is a query (SELECT statement) inside a query
- A subquery is normally expressed inside parentheses
- The first query in the SQL statement is known as the outer query
- The query inside the SQL statement is known as the inner query
- The inner query is executed first
- The output of an inner query is used as the input for the outer query
- The entire SQL statement is sometimes referred to as a nested query

A subquery can return one value or multiple values. To be precise, the subquery can return:

- *One single value (one column and one row)*. This subquery is used anywhere a single value is expected, as in the right side of a comparison expression. Obviously, when you assign a value to an attribute, that value is a single value, not a list of values. Therefore, the subquery must return only one value

(one column, one row). If the query returns multiple values, the DBMS will generate an error.

- *A list of values (one column and multiple rows).* This type of subquery is used anywhere a list of values is expected, such as when using the IN clause. This type of subquery is used frequently in combination with the IN operator in a WHERE conditional expression.
- *A virtual table (multicolumn, multirow set of values).* This type of subquery can be used anywhere a table is expected, such as when using the FROM clause.

It's important to note that a subquery can return no values at all; it is a NULL. In such cases, the output of the outer query may result in an error or a null empty set depending where the subquery is used (in a comparison, an expression, or a table set).

In the following sections, you will learn how to write subqueries within the SELECT statement to retrieve data from the database.

Note

You can also read more about subqueries in Chapter 9 Advanced SQL.

8.1 SELECT Subqueries

The most common type of subquery uses an inner SELECT subquery on the right side of a WHERE comparison expression. For example, to find the prices of all tickets with a price less than or equal to the average ticket price, you write the following query:

```

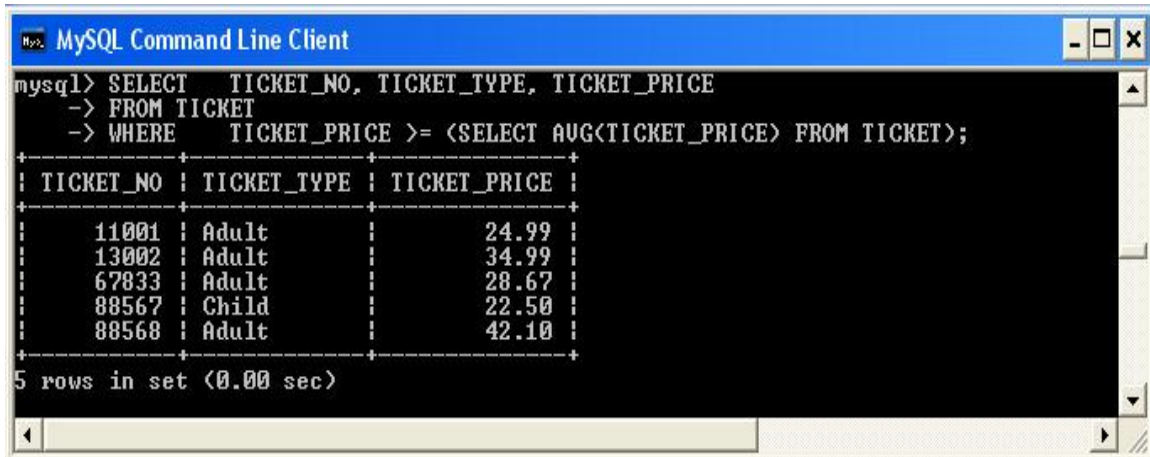
SELECT      TICKET_NO, TICKET_TYPE, TICKET_PRICE

FROM TICKET

WHERE      TICKET_PRICE >= (SELECT AVG(TICKET_PRICE) FROM TICKET);

```

The output of the query is shown in Figure 69.



```

mysql> SELECT      TICKET_NO, TICKET_TYPE, TICKET_PRICE
-> FROM TICKET
-> WHERE      TICKET_PRICE >= (SELECT AVG(TICKET_PRICE) FROM TICKET);
+-----+-----+-----+
| TICKET_NO | TICKET_TYPE | TICKET_PRICE |
+-----+-----+-----+
| 11001 | Adult | 24.99 |
| 13002 | Adult | 34.99 |
| 67833 | Adult | 28.67 |
| 88567 | Child | 22.50 |
| 88568 | Adult | 42.10 |
+-----+-----+-----+
5 rows in set (0.00 sec)

```

Figure 69 Example of SELECT Subquery

Note that this type of query, when used in a $>$, $<$, $=$, $>=$, or $<=$ conditional expression, requires a subquery that returns only one single value (one column, one row). The value generated by the subquery must be of a “comparable” data type; if the attribute to the left of the comparison symbol is a character type, the subquery must return a character string. Also, if the query returns more than a single value, the DBMS will generate an error.

Task 8.1 Write a query that displays the first name, last name of all employees who earn more than the average hourly rate. Do not display duplicate rows. Your output should match that shown in Figure 70.



```

mysql>
mysql>
mysql> SELECT DISTINCT(EMP_FNAME), EMP_LNAME
-> FROM EMPLOYEE NATURAL JOIN HOURS
-> WHERE HOUR_RATE > (SELECT AVG(HOUR_RATE) FROM HOURS);
+-----+-----+
| EMP_FNAME | EMP_LNAME |
+-----+-----+
| Enrica    | Denver    |
| Mirrelle  | Namowa    |
+-----+-----+
2 rows in set (0.08 sec)

mysql>

```

Figure 70 Output for task 8.1

8.2 IN Subqueries

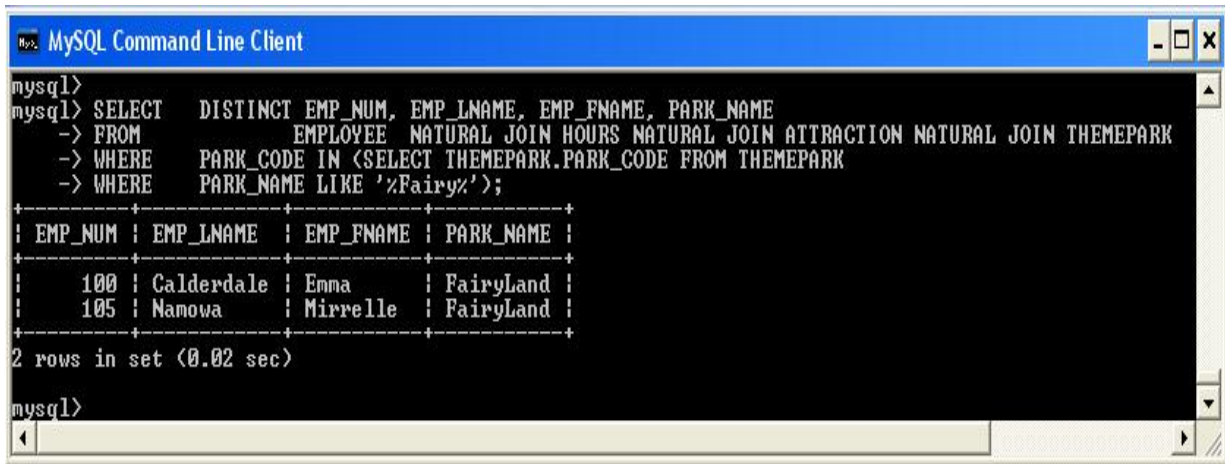
The following query displays all employees who work in a Theme Park that has the word 'Fairy' in its name. As there are a number of different Theme Parks that match this criteria you need to compare the PARK_CODE not to one park code (single value), but to a list of park codes. When you want to compare a single attribute to a list of values, you use the IN operator. When the PARK_CODE values are not known beforehand but they can be derived using a query, you must use an IN subquery. The following example lists all employees who have worked in such a Theme Park.

```

SELECT      DISTINCT EMP_NUM, EMP_LNAME, EMP_FNAME, PARK_NAME
FROM        EMPLOYEE NATURAL JOIN HOURS NATURAL JOIN
ATTRACTION NATURAL JOIN THEMEPARK
WHERE       PARK_CODE IN (SELECT THEMEPARK.PARK_CODE FROM
THEMEPARK WHERE      PARK_NAME LIKE '%Fairy%');

```

The result of that query is shown in Figure 71.



```

mysql>
mysql> SELECT DISTINCT EMP_NUM, EMP_LNAME, EMP_FNAME, PARK_NAME
-> FROM EMPLOYEE NATURAL JOIN HOURS NATURAL JOIN ATTRACTION NATURAL JOIN THEMEPARK
-> WHERE PARK_CODE IN (SELECT THEMEPARK.PARK_CODE FROM THEMEPARK
-> WHERE PARK_NAME LIKE '%Fairy%');
+-----+-----+-----+-----+
| EMP_NUM | EMP_LNAME | EMP_FNAME | PARK_NAME |
+-----+-----+-----+-----+
| 100 | Calderdale | Emma | FairyLand |
| 105 | Nanowa | Mirrelle | FairyLand |
+-----+-----+-----+-----+
2 rows in set (0.02 sec)

mysql>

```

Figure 71 Employees who work in a Theme Park LIKE ‘Fairy’.

Task 8.2 Enter and execute the above query and compare your output with that shown in Figure 71.

8.3 HAVING Subqueries

A subquery can also be used with a HAVING clause. Remember that the HAVING clause is used to restrict the output of a GROUP BY query by applying a conditional criteria to the grouped rows. For example, to list all PARK_CODES where the total quantity of tickets sold is greater than the average quantity sold, you would write the following query:

```

SELECT      PARK_CODE, SUM(LINE_QTY)

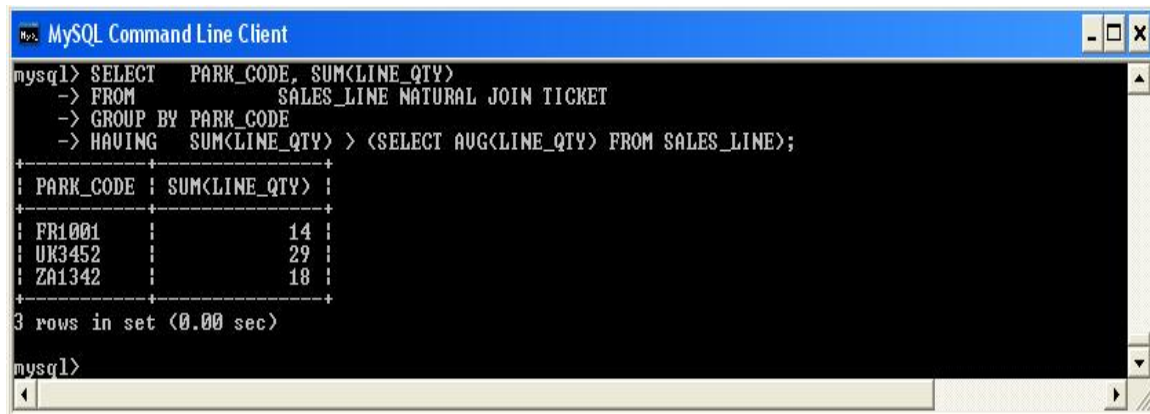
FROM        SALES_LINE NATURAL JOIN TICKET

GROUP BY PARK_CODE

HAVING SUM(LINE_QTY) > (SELECT AVG(LINE_QTY) FROM SALES_LINE);

```

The result of that query is shown in Figure 72.



```

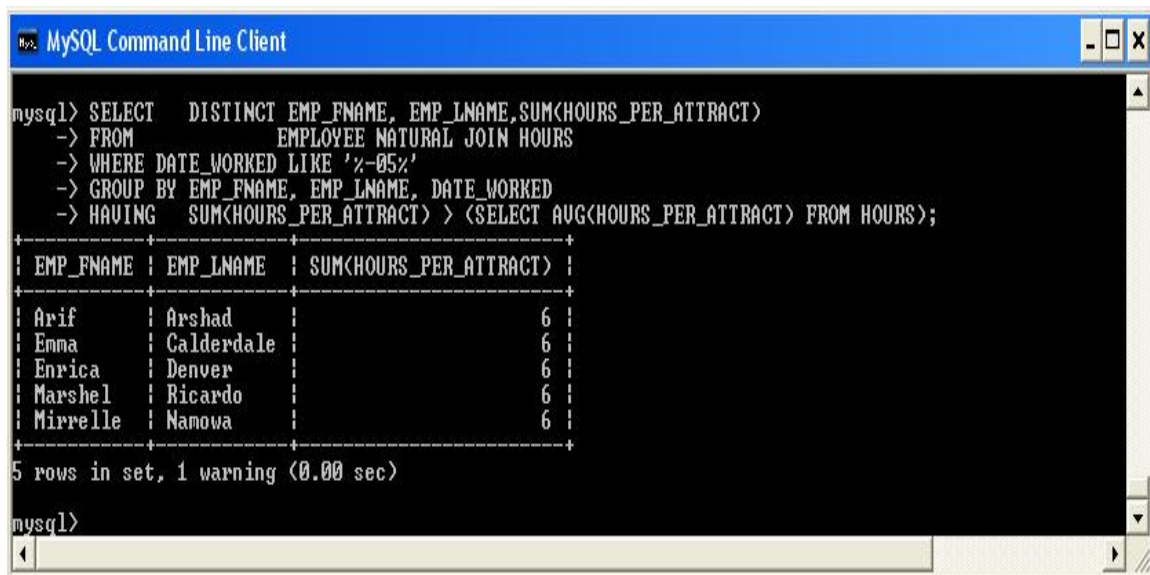
mysql> SELECT  PARK_CODE, SUM(LINE_QTY)
-> FROM      SALES_LINE NATURAL JOIN TICKET
-> GROUP BY  PARK_CODE
-> HAVING    SUM(LINE_QTY) > (SELECT AVG(LINE_QTY) FROM SALES_LINE);
+-----+-----+
| PARK_CODE | SUM(LINE_QTY) |
+-----+-----+
| FR1001    | 14            |
| UK3452    | 29            |
| ZA1342    | 18            |
+-----+-----+
3 rows in set (0.00 sec)

mysql>

```

Figure 72 PARK_CODES where tickets are selling above average.

Task 8.3 Using the query above as a guide, write a new query to display the first and last names of all employees who have worked in total less than the average number of hours in total during May 2007. Your output should match that shown in Figure 73.



```

mysql> SELECT  DISTINCT EMP_FNAME, EMP_LNAME, SUM(HOURS_PER_ATTRACT)
-> FROM      EMPLOYEE NATURAL JOIN HOURS
-> WHERE DATE_WORKED LIKE '%-05%'
-> GROUP BY  EMP_FNAME, EMP_LNAME, DATE_WORKED
-> HAVING    SUM(HOURS_PER_ATTRACT) > (SELECT AVG(HOURS_PER_ATTRACT) FROM HOURS);
+-----+-----+-----+
| EMP_FNAME | EMP_LNAME | SUM(HOURS_PER_ATTRACT) |
+-----+-----+-----+
| Arif      | Arshad    | 6                      |
| Emma      | Calderdale | 6                      |
| Enrica     | Denver    | 6                      |
| Marshal    | Ricardo    | 6                      |
| Mirrelle   | Namova     | 6                      |
+-----+-----+-----+
5 rows in set, 1 warning (0.00 sec)

mysql>

```

Figure 73 Output for task 8.3

8.4 Multirow Subquery operator ALL.

So far, you have learned that you must use an IN subquery when you need to compare a value to a list of values. But the IN subquery uses an equality operator; that is, it selects only those rows that match (are equal to) at least one of the values in the list. What happens if you need to do an inequality comparison (> or <) of one value to a list of values? For example, to find the ticket_numbers and corresponding park_codes of the tickets that are priced higher than the highest-priced 'Child' ticket you could write the following query.

```
SELECT TICKET_NO, PARK_CODE
FROM TICKET
WHERE TICKET_PRICE > ALL (SELECT TICKET_PRICE FROM TICKET
WHERE TICKET_TYPE = 'CHILD');
```

The output of that query is shown in Figure 74.



```
mysql> SELECT TICKET_NO, PARK_CODE
-> FROM TICKET
-> WHERE TICKET_PRICE > ALL (SELECT TICKET_PRICE FROM TICKET
-> WHERE TICKET_TYPE = 'CHILD');
+-----+-----+
| TICKET_NO | PARK_CODE |
+-----+-----+
| 11001 | SP4533 |
| 13002 | FR1001 |
| 67833 | ZA1342 |
| 88568 | UK3452 |
+-----+-----+
4 rows in set (0.00 sec)

mysql>
mysql>
```

Figure 74 Example of ALL.

This query is a typical example of a nested query. The use of the ALL operator allows you to compare a single value (TICKET_PRICE) with a list of values returned by the nested query, using a comparison operator other than equals. For a row to appear in the result set, it has to meet the criterion `TICKET_PRICE > ALL` of the individual values returned by the nested query.

8.5 Attribute list Subqueries

The SELECT statement uses the attribute list to indicate what columns to project in the resulting set. Those columns can be attributes of base tables or computed attributes or the result of an aggregate function. The attribute list can also include a subquery expression, also known as an inline subquery. A subquery in the attribute list must return one single value; otherwise, an error code is raised. For example, a simple inline query can be used to list the difference between each tickets' price and the average ticket price:

```
SELECT      TICKET_NO, TICKET_PRICE,

            (SELECT AVG(TICKET_PRICE) FROM TICKET) AS AVGPRICE,

            TICKET_PRICE - (SELECT AVG(TICKET_PRICE) FROM TICKET) AS DIFF

FROM TICKET;
```

The output for this query is shown in Figure 75.


```

mysql> SELECT  TICKET_NO, TICKET_PRICE,
-> <SELECT AVG(TICKET_PRICE) FROM TICKET> AS AUGPRICE, TICKET_PRICE - <SELECT AVG(TICKET_PRICE) FROM T
-> FROM      TICKET;
+-----+-----+-----+-----+
| TICKET_NO | TICKET_PRICE | AUGPRICE | DIFF |
+-----+-----+-----+-----+
| 11001 | 24.99 | 21.740000 | 3.250000 |
| 11002 | 14.99 | 21.740000 | -6.750000 |
| 11003 | 10.99 | 21.740000 | -10.750000 |
| 13001 | 18.99 | 21.740000 | -2.750000 |
| 13002 | 34.99 | 21.740000 | 13.250000 |
| 13003 | 20.99 | 21.740000 | -0.750000 |
| 67832 | 18.56 | 21.740000 | -3.180000 |
| 67833 | 28.67 | 21.740000 | 6.930000 |
| 67855 | 12.12 | 21.740000 | -9.620000 |
| 88567 | 22.50 | 21.740000 | 0.760000 |
| 88568 | 42.10 | 21.740000 | 20.360000 |
| 89720 | 10.99 | 21.740000 | -10.750000 |
+-----+-----+-----+-----+
12 rows in set (0.00 sec)

mysql> _

```

Figure 75 Displaying the difference in ticket prices.

This inline query output returns one single value (the average ticket's price) and that the value is the same in every row. Note also that the query used the full expression instead of the column aliases when computing the difference. In fact, if you try to use the alias in the difference expression, you will get an error message. The column alias cannot be used in computations in the attribute list when the alias is defined in the same attribute list.

Task 8.4 Write a query to display an employee's first name, last name and date worked which lists the difference between the number of hours an employee has worked on an attraction and the average hours worked on that attraction. Label this column 'DIFFERENCE' and the average hours column 'AVERAGE'.

8.6 Correlated Subqueries

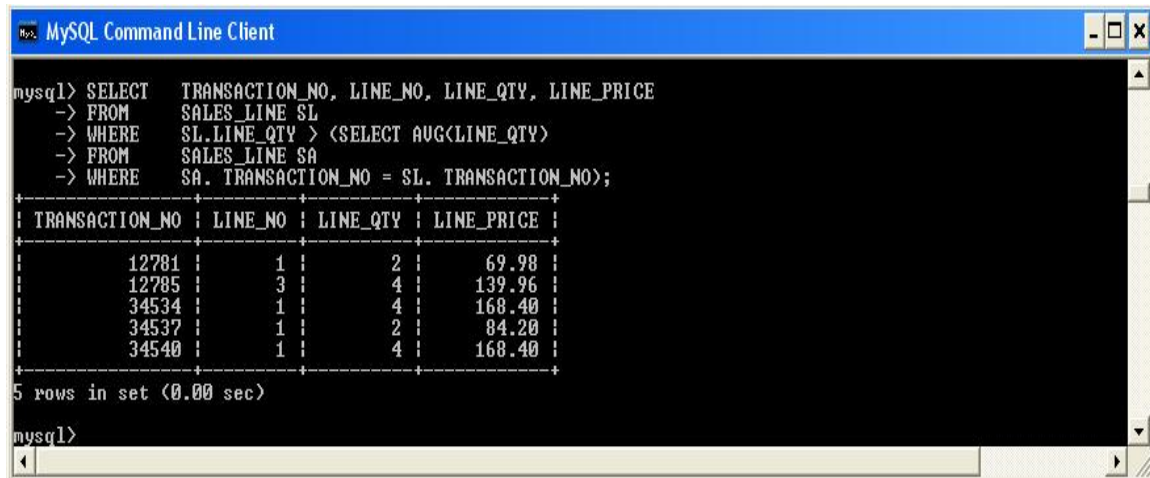
A correlated subquery is a subquery that executes once for each row in the outer query.

The relational DBMS uses the same sequence to produce correlated subquery results:

1. It initiates the outer query.
2. For each row of the outer query result set, it executes the inner query by passing the outer row to the inner query.

That process is the opposite of the subqueries you have seen so far. The query is called a *correlated* subquery because the inner query is *related* to the outer query because the inner query references a column of the outer subquery. For example, suppose you want to know all the ticket sales in which the quantity sold value is greater than the average quantity sold value for *that* ticket (as opposed to the average for *all tickets*). The following correlated query completes the preceding two-step process:

```
SELECT    TRANSACTION_NO, LINE_NO, LINE_QTY, LINE_PRICE
FROM      SALES_LINE SL
WHERE     SL.LINE_QTY > (SELECT AVG(LINE_QTY)
FROM      SALES_LINE SA
WHERE     SA. TRANSACTION_NO = SL. TRANSACTION_NO);
```



```

mysql> SELECT TRANSACTION_NO, LINE_NO, LINE_QTY, LINE_PRICE
-> FROM SALES_LINE SL
-> WHERE SL.LINE_QTY > (SELECT AVG(LINE_QTY)
-> FROM SALES_LINE SA
-> WHERE SA.TRANSACTION_NO = SL.TRANSACTION_NO);
+-----+-----+-----+-----+
| TRANSACTION_NO | LINE_NO | LINE_QTY | LINE_PRICE |
+-----+-----+-----+-----+
| 12781 | 1 | 2 | 69.98 |
| 12785 | 3 | 4 | 139.96 |
| 34534 | 1 | 4 | 168.40 |
| 34537 | 1 | 2 | 84.20 |
| 34540 | 1 | 4 | 168.40 |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>

```

Figure 76 Example of a correlated subquery

As you examine the output shown in figure 76, note that the SALES_LINE table is used more than once; so you must use table aliases.

Correlated subqueries can also be used with the EXISTS special operator. For example, suppose you want to know all the names of all Theme Parks where tickets have been recently sold. In that case, you could use a correlated subquery as follows:

```

SELECT      PARK_CODE, PARK_NAME, PARK_COUNTRY

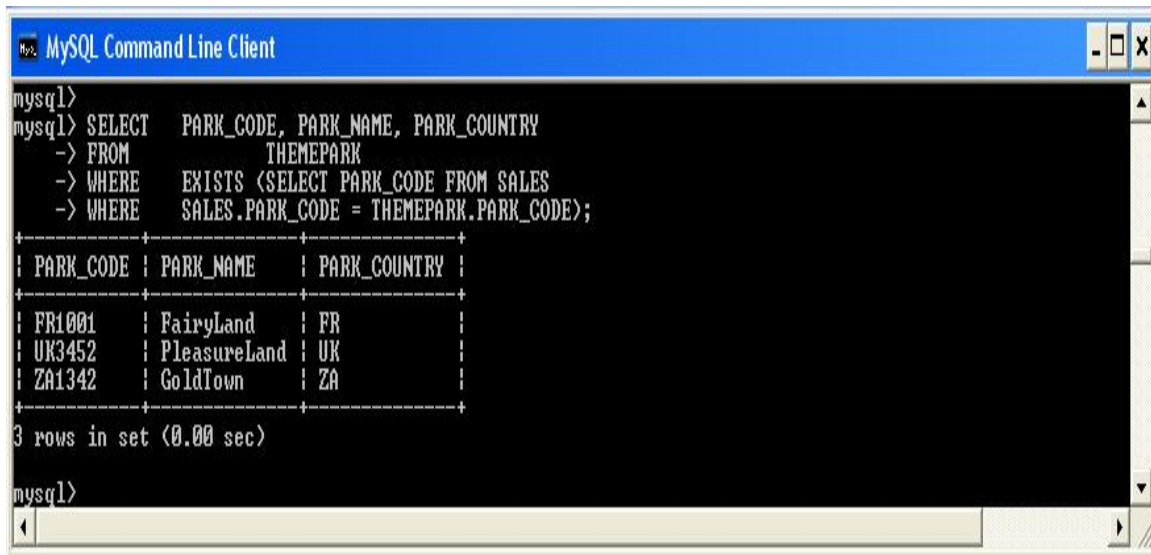
FROM        THEMEPARK

WHERE       EXISTS (SELECT PARK_CODE FROM SALES

WHERE       SALES.PARK_CODE = THEMEPARK.PARK_CODE);

```

The output for this query is shown in figure 77.



```
mysql>
mysql> SELECT  PARK_CODE, PARK_NAME, PARK_COUNTRY
-> FROM      THEMEPARK
-> WHERE     EXISTS (SELECT PARK_CODE FROM SALES
-> WHERE     SALES.PARK_CODE = THEMEPARK.PARK_CODE);

+-----+-----+-----+
| PARK_CODE | PARK_NAME | PARK_COUNTRY |
+-----+-----+-----+
| FR1001    | FairyLand | FR           |
| UK3452    | PleasureLand | UK          |
| ZA1342    | GoldTown  | ZA           |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Figure 77 Example of correlated subqueries

Task 8.5 Type in and execute the two correlated subqueries in this section and check your output against that shown in figures 76 and 77.

Task 8.6 Modify the second query you entered in task 8.5 to display all the theme parks where there have been no recorded tickets sales recently.