

Database Systems Lab



Lab # 06

Joining Database Tables and Aggregate functions

Instructor: Engr. Muhammad Usman

Email: usman.rafiq@nu.edu.pk

Course Code: CL2005

Semester Fall 2021

Department of Computer Science,

National University of Computer and Emerging Sciences FAST
Peshawar Campus

Contents

1. Aggregate Functions.....	2
Table 1 Basic SQL Aggregate Functions.....	2
COUNT.....	2
MAX and MIN.....	5

SUM and AVG.....	6
GROUP BY.....	6
HAVING	8
2. Introduction to Joins.....	9
2.1. Joining tables with an alias.....	13
Natural Join	14
Join USING.....	15
ON	16
Outer Join.....	18
Join	21
Exercises.....	22

1. Aggregate Functions

SQL can perform mathematical summaries through the use of aggregate (or group) functions, such as counting the number of rows that contain a specified condition,

finding the minimum or maximum values for a specified attribute, summing the values in a specified column, and averaging the values in a specified column.

Aggregate functions return results based on groups of rows. By default, the entire result is treated as one group.

Table 3 shows some of the basic aggregate functions.

Table 1 Basic SQL Aggregate Functions

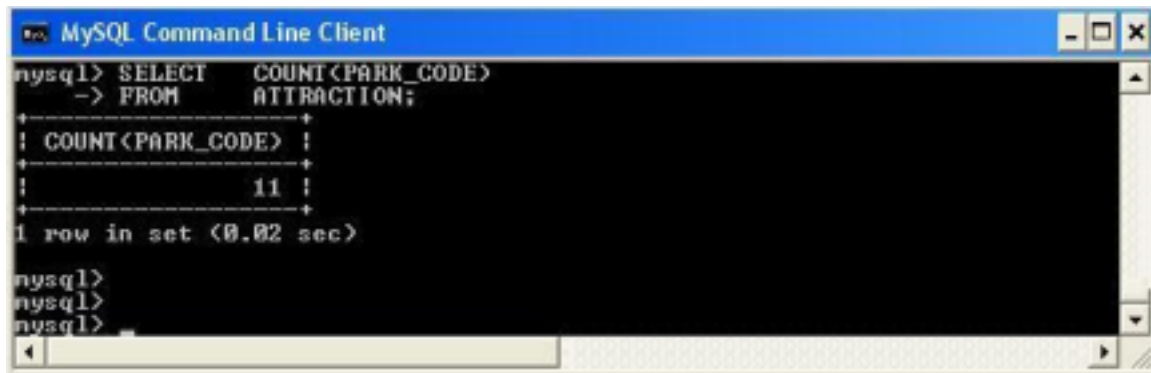
FUNCTION	OUTPUT
COUNT	The number of rows containing non-null values
MIN	The minimum attribute value encountered in a given column
MAX	The maximum attribute value encountered in a given column
SUM	The sum of all values for a given column
AVG	The arithmetic mean (average) for a specified column

COUNT

The COUNT function is used to tally the number of non-null values of an attribute. COUNT can be used in conjunction with the DISTINCT clause. If you wanted to find out how many different theme parks contained attractions from the ATTRACTION table you would write the following query:

```
SELECT COUNT(PARK_CODE)
FROM ATTRACTION;
```

The query would return 11 rows as shown in Figure 37.



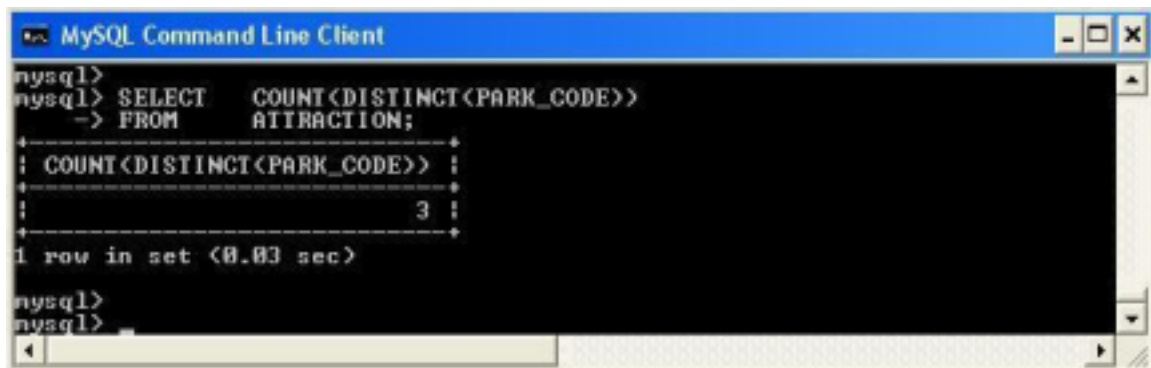
```
mysql> SELECT COUNT(PARK_CODE)
-> FROM ATTRACTION;
+-----+
| COUNT(PARK_CODE) |
+-----+
| 11                |
+-----+
1 row in set (0.02 sec)

mysql>
mysql>
mysql>
```

Figure 37: Counting the number of Theme parks in ATTRACTION

However, if you wanted to know how many different Theme parks were in the ATTRACTION table, you would modify the query as follows (For the output see Figure 38):

```
SELECT COUNT (DISTINCT (PARK_CODE ) )
FROM ATTRACTION;
```

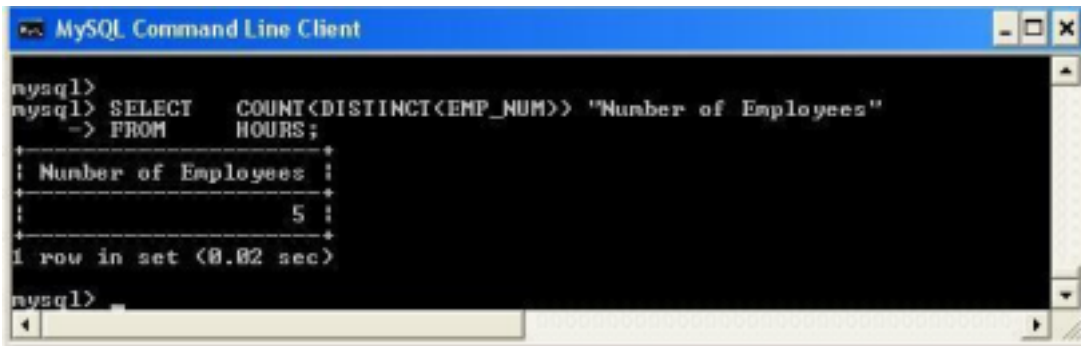


```
mysql> SELECT COUNT(DISTINCT(PARK_CODE))
-> FROM ATTRACTION;
+-----+
| COUNT(DISTINCT(PARK_CODE)) |
+-----+
| 3                            |
+-----+
1 row in set (0.03 sec)

mysql>
mysql>
```

Figure 38: Counting the number of DISTINCT Theme parks in ATTRACTION.

Task 1 Write a query that displays the number of distinct employees in the HOURS table. You should label the column “Number of Employees”. Your output should match that shown in Figure 39.



```
mysql>
mysql> SELECT      COUNT(DISTINCT(EMP_NUM)) "Number of Employees"
-> FROM      HOURS;

+-----+
| Number of Employees |
+-----+
|                    5 |
+-----+
1 row in set (0.02 sec)

mysql>
```

Figure 39: Query output for Task 1

COUNT always returns the number of non-null values in the given column. Another use for the COUNT function is to display the number of rows returned by a query using the syntax COUNT(*).

Task 2 Enter the following two queries and examine their output shown in Figure 40.

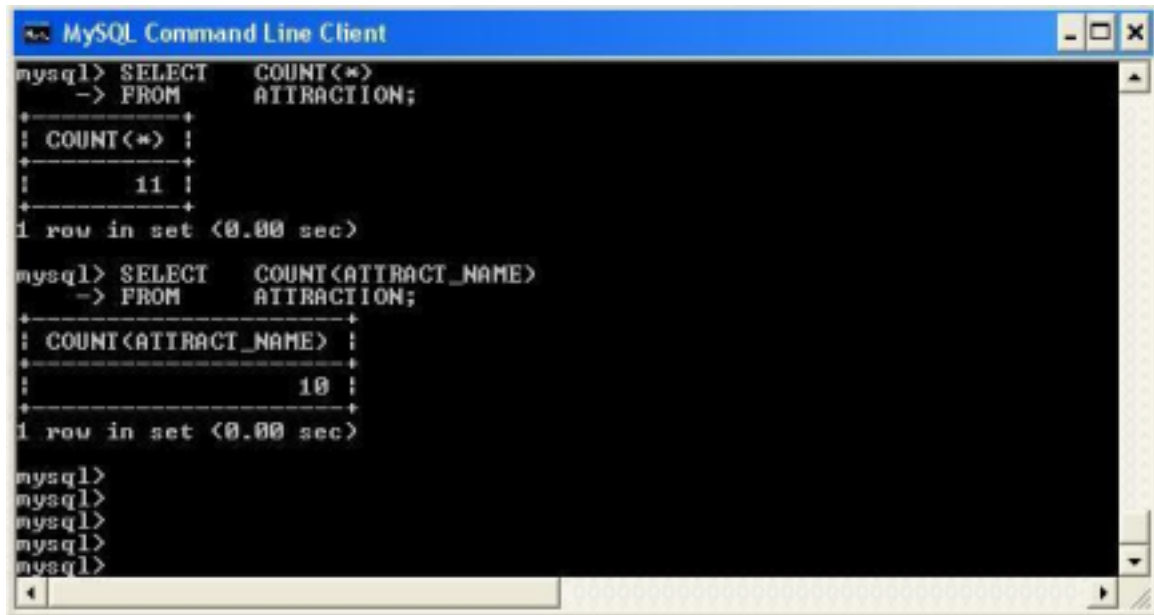
Can you explain why the number of rows returned is different?

```
SELECT COUNT(*)
```

```
FROM ATTRACTION;
```

```
SELECT COUNT(ATTRACT_NAME)
```

```
FROM ATTRACTION;
```



```
mysql> SELECT COUNT(*)
-> FROM ATTRACTION;
+-----+
| COUNT(*) |
+-----+
|      11 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(ATTRACT_NAME)
-> FROM ATTRACTION;
+-----+
| COUNT(ATTRACT_NAME) |
+-----+
|          10 |
+-----+
1 row in set (0.00 sec)

mysql>
mysql>
mysql>
mysql>
mysql>
```

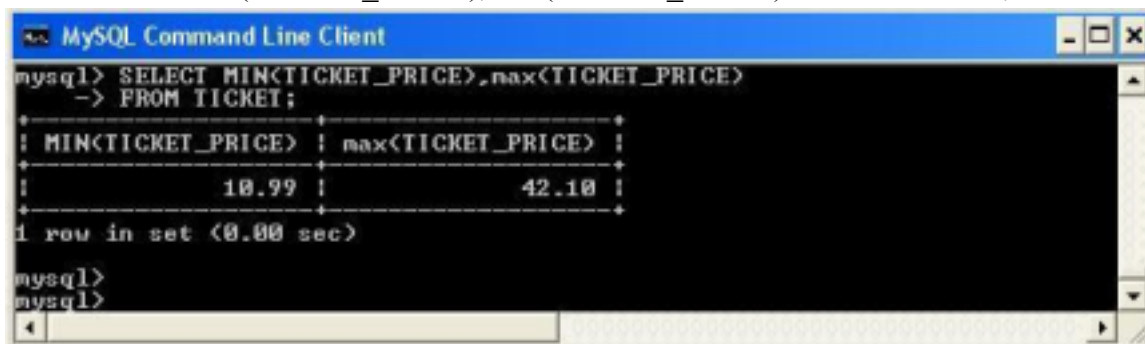
Figure 40: Examples of using the COUNT function

MAX and MIN

The MAX and MIN functions are used to find answers to problems such as What is the highest and lowest ticket price sold in all Theme parks.

Task 3 Enter the following query which illustrates the use of the MIN and Max functions. Check the query results with those shown in Figure 41.

SELECT MIN(TICKET_PRICE),max(TICKET_PRICE) FROM TICKET;



```
mysql> SELECT MIN(TICKET_PRICE),max(TICKET_PRICE)
-> FROM TICKET;
+-----+-----+
| MIN(TICKET_PRICE) | max(TICKET_PRICE) |
+-----+-----+
|          10.99 |          42.10 |
+-----+-----+
1 row in set (0.00 sec)

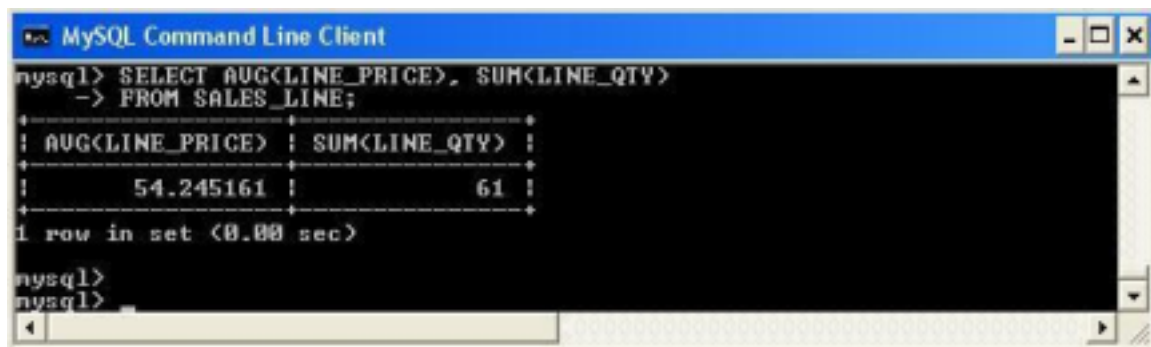
mysql>
mysql>
```

Figure 41: Examples of using the MIN and MAX functions

SUM and AVG

The SUM function computes the total sum for any specified attribute, using whatever condition(s) you have imposed. The AVG function calculates the arithmetic mean (average) for a specified attribute. The following query displays the average amount spent on Theme park tickets per customer (LINE_PRICE) and the total number of tickets purchase (LINE_QTY). Figure 42 shows the output for this query.

```
SELECT AVG(LINE_PRICE), SUM(LINE_QTY)
FROM SALES_LINE;
```



```
mysql> SELECT AVG(LINE_PRICE), SUM(LINE_QTY)
-> FROM SALES_LINE;
+-----+-----+
| AVG(LINE_PRICE) | SUM(LINE_QTY) |
+-----+-----+
| 54.245161       | 61            |
+-----+-----+
1 row in set (0.00 sec)

mysql>
mysql>
```

Figure 42: Example showing the AVG and SUM functions

GROUP BY

The GROUP BY clause is generally used when you have attribute columns combined with aggregate functions in the SELECT statement. It is valid only when used in conjunction with one of the SQL aggregate functions, such as COUNT, MIN, MAX, AVG and SUM. The GROUP BY clause appears after the WHERE statement. When using GROUP BY you should include all the attributes that are in the SELECT statement that do not use an aggregate function. The syntax is

```
SELECT columnlist
FROM tablelist
[WHERE conditionlist ]
[GROUP BY columnlist ]
[HAVING conditionlist ]
[ORDER BY columnlist [ASC | DESC] ];
```

The following query displays the minimum and maximum ticket price of all parks. The output is shown in Figure 43. Notice that the query groups only by the PARK_CODE as no aggregate function is applied to this attribute in the SELECT statement.

```
SELECT PARK_CODE, MIN(TICKET_PRICE),MAX(TICKET_PRICE)
FROM TICKET
GROUP BY PARK_CODE;
```



The screenshot shows a MySQL Command Line Client window. The user has entered the following query:

```
mysql> SELECT PARK_CODE, MIN(TICKET_PRICE),MAX(TICKET_PRICE)
mysql> -> FROM TICKET
mysql> -> GROUP BY PARK_CODE;
```

The output is displayed as a table with 4 rows and 3 columns:

PARK_CODE	MIN(TICKET_PRICE)	MAX(TICKET_PRICE)
FR1001	18.99	34.99
SP4533	10.99	24.99
UK3452	10.99	42.10
ZA1342	12.12	28.67

Below the table, it says "4 rows in set (0.00 sec)".

Figure 43: Displaying minimum and maximum ticket prices for each PARK_CODE

43. What happens if you miss out the GROUP BY clause?

HAVING

The HAVING clause is an extension to the GROUP BY clause and is applied to the output of a GROUP BY operation. The HAVING clause operates very much like the WHERE clause in the SELECT statement. However, the WHERE clause applies to columns and expressions for individual rows, while the HAVING clause is applied to the output of a GROUP BY operation.

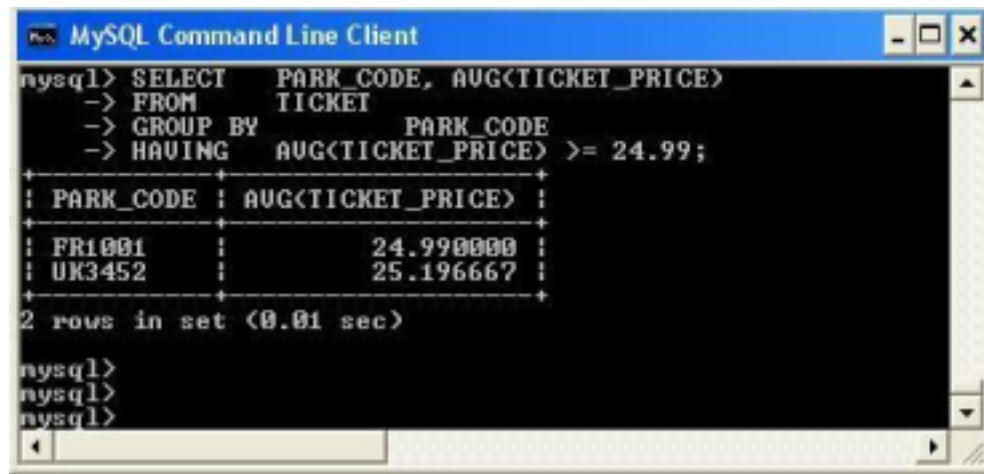
Supposing you wanted to list the average ticket price at each Theme Park but wanted to limit the listing to Theme Parks whose average ticket price was greater or equal to €24.99. This can be achieved by the following query whose output is shown in Figure 44.

```
SELECT PARK_CODE, AVG(TICKET_PRICE)
```

```
FROM TICKET
```

```
GROUP BY PARK_CODE
```

```
HAVING AVG(TICKET_PRICE) >= 24.99;
```



```
mysql> SELECT PARK_CODE, AVG(TICKET_PRICE)
-> FROM
-> GROUP BY PARK_CODE
-> HAVING AVG(TICKET_PRICE) >= 24.99;
+-----+-----+
| PARK_CODE | AVG(TICKET_PRICE) |
+-----+-----+
| FR1001    | 24.990000         |
| UK3452    | 25.196667         |
+-----+-----+
2 rows in set (0.01 sec)

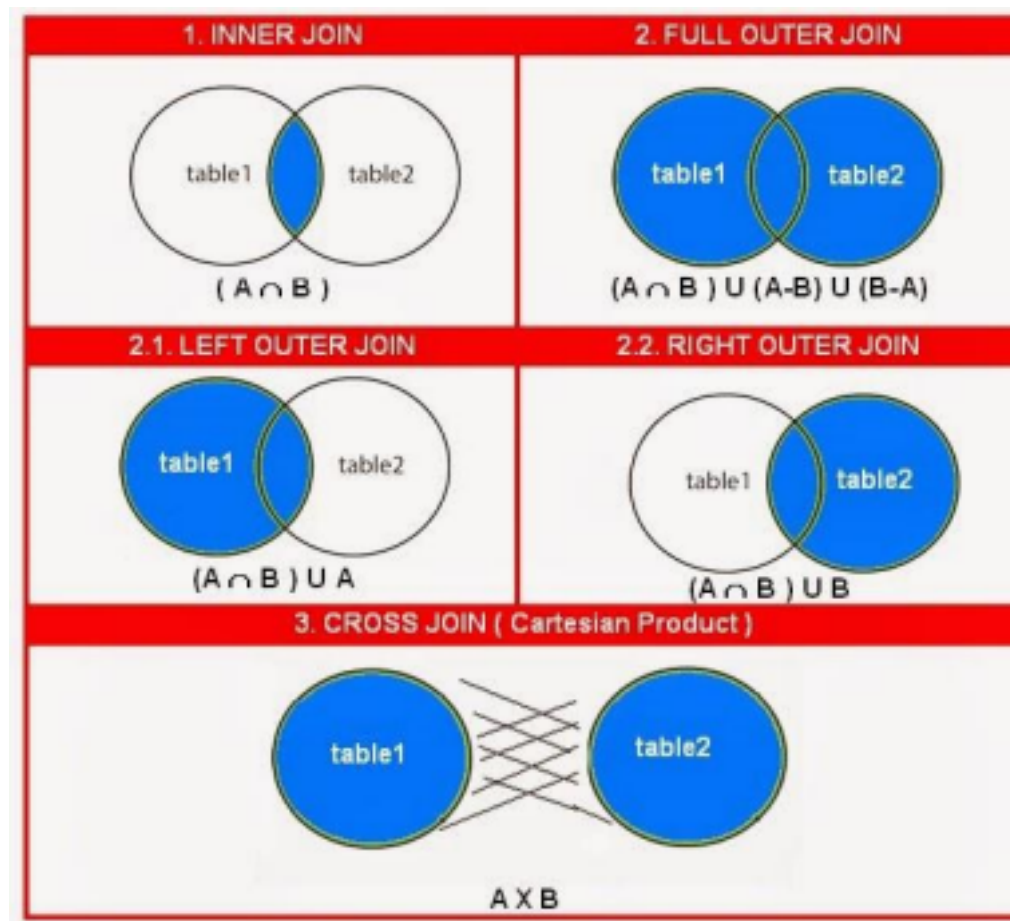
mysql>
mysql>
mysql>
```

Figure 44: Example of the HAVING clause

2. Introduction to Joins

The relational join operation merges rows from two or more tables and returns the rows with one of the following conditions:

- Have common values in common columns (natural join)
- Meet a given join condition (equality or inequality)
- Have common values in common columns or have no matching values (outer join)



There are a number of different joins that can be performed. The most common is the natural join. To join tables, you simply enumerate the tables in the FROM clause of the SELECT statement. The DBMS will create the Cartesian product of every table in the FROM clause. However, to get the correct result—that is, a natural join—you must select only the rows in which the common attribute values match. That is done with the WHERE clause. Use the WHERE clause to indicate the common attributes that are used to link the tables (sometimes referred to as the *join condition*). For example, suppose you want to join the two tables THEMEPARK and TICKET. Because PARK_CODE is the foreign key in the TICKET table and the primary key in the THEMEPARK table, the link is established on PARK_CODE. It is important to note that when the same attribute name appears in more than one of the joined tables, the source table of the attributes

Page | 10

listed in the SELECT command sequence must be defined. To join the THEMEPARK and TICKET tables, you would use the following, which produces the output shown in

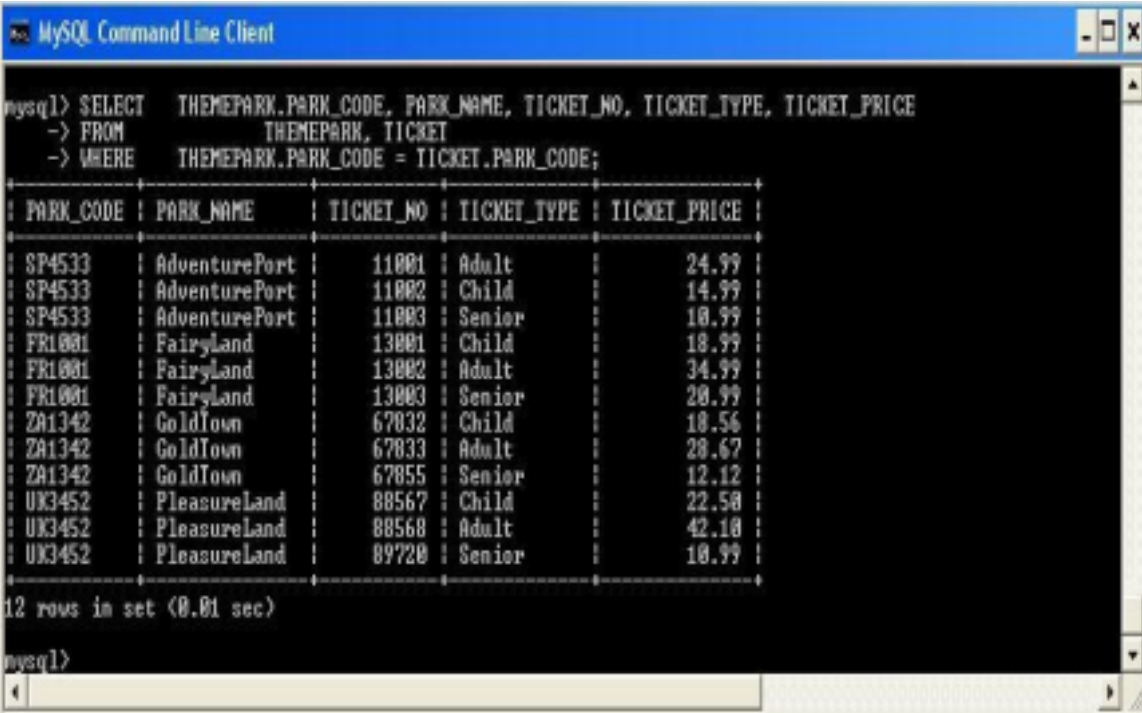
Figure 46.

```
SELECT THEMEPARK.PARK_CODE, PARK_NAME, TICKET_NO,
```

```
TICKET_TYPE, TICKET_PRICE
```

```
FROM THEMEPARK, TICKET
```

```
WHERE THEMEPARK.PARK_CODE = TICKET.PARK_CODE
```



The screenshot shows a MySQL Command Line Client window with a black background and white text. The query entered is:
mysql> SELECT THEMEPARK.PARK_CODE, PARK_NAME, TICKET_NO, TICKET_TYPE, TICKET_PRICE
-> FROM THEMEPARK, TICKET
-> WHERE THEMEPARK.PARK_CODE = TICKET.PARK_CODE;
The results are displayed in a table with 5 columns: PARK_CODE, PARK_NAME, TICKET_NO, TICKET_TYPE, and TICKET_PRICE. There are 12 rows of data. Below the table, it says "12 rows in set (0.01 sec)". The prompt "mysql>" is visible at the bottom.

PARK_CODE	PARK_NAME	TICKET_NO	TICKET_TYPE	TICKET_PRICE
SP4533	AdventurePort	11001	Adult	24.99
SP4533	AdventurePort	11002	Child	14.99
SP4533	AdventurePort	11003	Senior	18.99
FR1001	FairyLand	13001	Child	18.99
FR1001	FairyLand	13002	Adult	34.99
FR1001	FairyLand	13003	Senior	20.99
ZA1342	GoldTown	67032	Child	10.56
ZA1342	GoldTown	67033	Adult	20.67
ZA1342	GoldTown	67055	Senior	12.12
UX3452	PleasureLand	88567	Child	22.50
UX3452	PleasureLand	88568	Adult	42.10
UX3452	PleasureLand	89720	Senior	10.99

Figure 46: Natural Join between THEMEPARK and TICKET tables

As you examine the preceding query, note the following points:

- The FROM clause indicates which tables are to be joined. If three or more tables are

included, the join operation takes place two tables at a time, starting from left to right.

For example, if you are joining tables T1, T2, and T3, first table T1 is joined to T2; the results of that join are then joined to table T3.

- The join condition in the WHERE clause tells the SELECT statement which rows will be returned. In this case, the SELECT statement returns all rows for which the PARK_CODE values in the THEMEPARK and TICKET tables are equal.
- The number of join conditions is always equal to the number of tables being joined minus one. For example, if you join three tables (T1, T2, and T3), you will have two join conditions (j1 and j2). All join conditions are connected through an AND logical operator. The first join condition (j1) defines the join criteria for T1 and T2. The second join condition (j2) defines the join criteria for the output of the first join and table T3.
- Generally, the join condition will be an equality comparison of the primary key in one table and the related foreign key in the second table.

Task 5 Execute the following query and check your results with those shown in Figure 47. Then modify the SELECT statement and change THEMEPARK.PARK_CODE to just PARK_CODE. What happens?

```
SELECT THEMEPARK.PARK_CODE, PARK_NAME, ATTRACT_NAME,  
        ATTRACT_CAPACITY  
FROM THEMEPARK, ATTRACTION  
        WHERE THEMEPARK.PARK_CODE = ATTRACTION.PARK_CODE;
```

Page | 12

```

mysql> SELECT THEMEPARK.PARK_CODE, PARK_NAME, ATTRACT_NAME, ATTRACT_CAPACITY
-> FROM THEMEPARK, ATTRACTION
-> WHERE THEMEPARK.PARK_CODE = ATTRACTION.PARK_CODE;
+-----+-----+-----+-----+
| PARK_CODE | PARK_NAME | ATTRACT_NAME | ATTRACT_CAPACITY |
+-----+-----+-----+-----+
| FR1001 | FairyLand | ThunderCoaster | 34 |
| FR1001 | FairyLand | SpinningTeacups | 62 |
| FR1001 | FairyLand | FlightToStars | 24 |
| FR1001 | FairyLand | Ant-Trap | 30 |
| ZA1342 | GoldTown | NULL | 40 |
| FR1001 | FairyLand | Carnival | 120 |
| UX3452 | PleasureLand | 3D-Lego_Show | 200 |
| UX3452 | PleasureLand | BlackHole2 | 34 |
| UX3452 | PleasureLand | Pirates | 42 |
| UX3452 | PleasureLand | UnderSeaWord | 80 |
| ZA1342 | GoldTown | GoldRush | 80 |
+-----+-----+-----+-----+
11 rows in set (0.00 sec)

mysql>
mysql>

```

Figure 47: Query output for task 5

2.1. Joining tables with an alias

An alias may be used to identify the source table from which the data are taken. For example, the aliases P and T can be used to label the THEMEPARK and TICKET tables as shown in the query below (which produces the same output as shown in Figure 46). Any legal table name may be used as an alias.

```

SELECT P.PARK_CODE, PARK_NAME, TICKET_NO, TICKET_TYPE,
       TICKET_PRICE
FROM THEMEPARK P, TICKET T
WHERE P.PARK_CODE =T.PARK_CODE;

```

2.2. Natural Join

The natural join returns all rows with matching values in the matching columns and eliminates duplicate columns. That style of query is used when the tables share one

or more common attributes with common names. The natural join syntax is:

```
SELECT column-list FROM table1 NATURAL JOIN table2
```

The natural join will perform the following tasks:

- Determine the common attribute(s) by looking for attributes with identical names and compatible data types
- Select only the rows with common values in the common attribute(s) •

If there are no common attributes, return the relational product of the two tables

The following example performs a natural join of the SALES and SALES_LINE tables and returns only selected attributes:

```
SELECT TRANSACTION_NO, SALE_DATE, LINE_NO, LINE_QTY,  
        LINE_PRICE  
FROM SALES NATURAL JOIN SALES_LINE;
```

The results of this query can be seen in Figure 48.

```

mysql> SELECT TRANSACTION_NO, SALE_DATE, LINE_NO, LINE_QTY,
-> LINE_PRICE
-> FROM SALES NATURAL JOIN SALES_LINE;

```

TRANSACTION_NO	SALE_DATE	LINE_NO	LINE_QTY	LINE_PRICE
12781	2007-05-18	1	2	69.98
12781	2007-05-18	2	1	14.99
12782	2007-05-18	1	2	69.98
12783	2007-05-18	1	2	41.98
12784	2007-05-18	2	1	14.99
12785	2007-05-18	1	1	14.99
12785	2007-05-18	2	1	34.99
12785	2007-05-18	3	4	139.96
34534	2007-05-18	1	4	168.40
34534	2007-05-18	2	1	22.50
34534	2007-05-18	3	2	21.98
34535	2007-05-18	1	2	84.20
34536	2007-05-18	1	2	21.98
34537	2007-05-18	1	2	84.20
34537	2007-05-18	2	1	22.50
34538	2007-05-18	1	2	21.98
34539	2007-05-18	1	2	21.98
34539	2007-05-18	2	2	84.20
34540	2007-05-18	1	4	168.40
34540	2007-05-18	2	1	22.50
34540	2007-05-18	3	2	21.98
34541	2007-05-18	1	2	84.20
67589	2007-05-18	1	2	57.34
67589	2007-05-18	2	2	37.12
67590	2007-05-18	1	2	57.34
67590	2007-05-18	2	2	37.12
67591	2007-05-18	1	1	18.56
67591	2007-05-18	2	1	12.12
67592	2007-05-18	1	4	114.68
67593	2007-05-18	1	2	57.34
67593	2007-05-18	2	2	37.12

```

31 rows in set (0.00 sec)

mysql>
mysql>
mysql>

```

Figure 48: Results of SALES NATURAL JOIN SALES_LINE;

One important difference between the natural join and the “old-style” join syntax as illustrated in Figure 46, Section 2.3, is that the NATURAL JOIN command does not require the use of a table qualifier for the common attributes.

2.3. Join USING

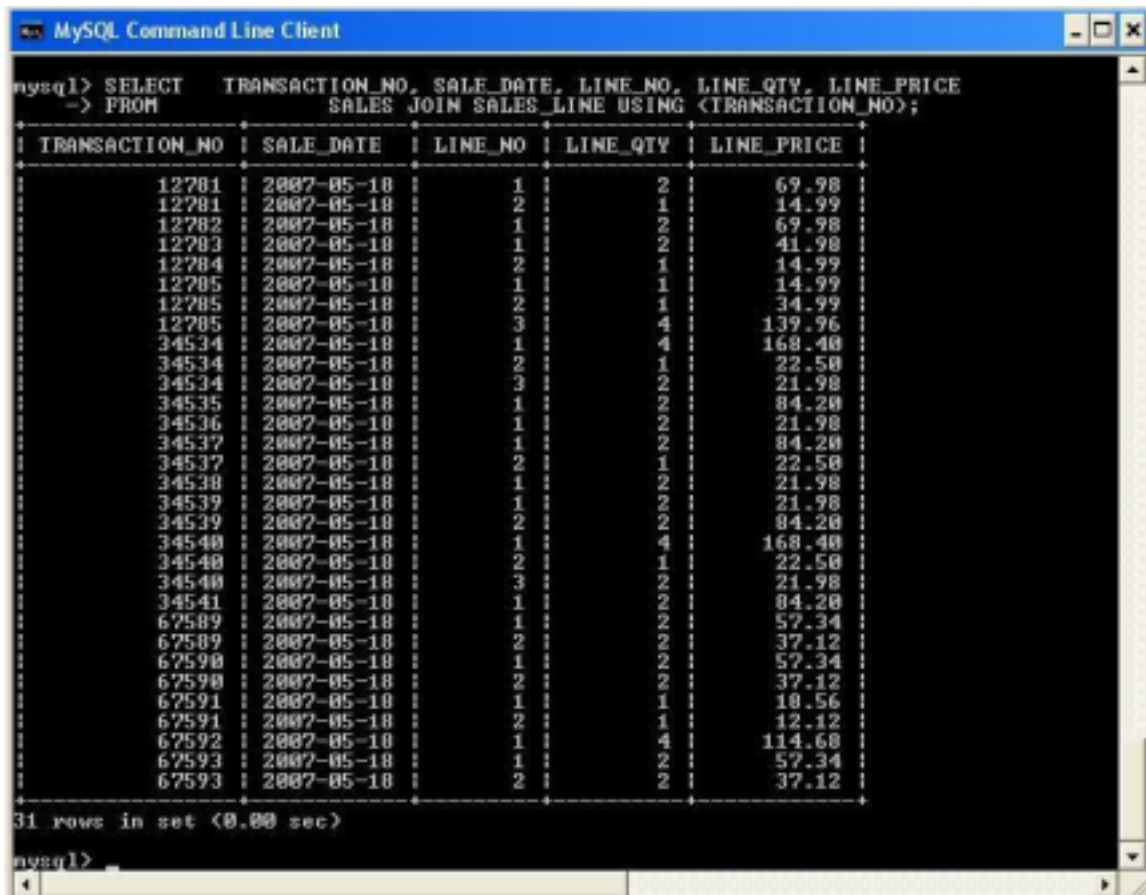
A second way to express a join is through the USING keyword. That query returns only the rows with matching values in the column indicated in the USING clause—and that column must exist in both tables. The syntax is:

```
SELECT column-list FROM table1 JOIN table2 USING (common-column)
```


To see the JOIN USING query in action, let's perform a join of the SALES and SALES_LINE tables by writing:

```
SELECT TRANSACTION_NO, SALE_DATE, LINE_NO, LINE_QTY,
       LINE_PRICE
FROM SALES JOIN SALES_LINE USING (TRANSACTION_NO);
```

The SQL statement produces the results shown in Figure 50.



The screenshot shows a MySQL Command Line Client window with the following query and results:

```
mysql> SELECT TRANSACTION_NO, SALE_DATE, LINE_NO, LINE_QTY, LINE_PRICE
-> FROM SALES JOIN SALES_LINE USING (TRANSACTION_NO);
```

TRANSACTION_NO	SALE_DATE	LINE_NO	LINE_QTY	LINE_PRICE
12781	2007-05-18	1	2	69.98
12781	2007-05-18	2	1	14.99
12782	2007-05-18	1	2	69.98
12783	2007-05-18	1	2	41.98
12784	2007-05-18	2	1	14.99
12785	2007-05-18	1	1	14.99
12785	2007-05-18	2	1	34.99
12785	2007-05-18	3	4	139.96
34534	2007-05-18	1	4	168.40
34534	2007-05-18	2	1	22.50
34534	2007-05-18	3	2	21.98
34535	2007-05-18	1	2	84.20
34536	2007-05-18	1	2	21.98
34537	2007-05-18	1	2	84.20
34537	2007-05-18	2	1	22.50
34538	2007-05-18	1	2	21.98
34539	2007-05-18	1	2	21.98
34539	2007-05-18	2	2	84.20
34540	2007-05-18	1	4	168.40
34540	2007-05-18	2	1	22.50
34540	2007-05-18	3	2	21.98
34541	2007-05-18	1	2	84.20
67589	2007-05-18	1	2	57.34
67589	2007-05-18	2	2	37.12
67590	2007-05-18	1	2	57.34
67590	2007-05-18	2	2	37.12
67591	2007-05-18	1	1	10.56
67591	2007-05-18	2	1	12.12
67592	2007-05-18	1	4	114.60
67593	2007-05-18	1	2	57.34
67593	2007-05-18	2	2	37.12

31 rows in set (0.00 sec)

Figure 50: Query results for SALES JOIN SALES_LINE USING TRANSACTION_NO

As was the case with the NATURAL JOIN command, the JOIN USING operand does not require table qualifiers.

2.4. Join ON

The previous two join styles used common attribute names in the joining tables. Another way to express a join when the tables have no common attribute names is to use the JOIN ON operand. That query will return only the rows that meet the indicated join condition.

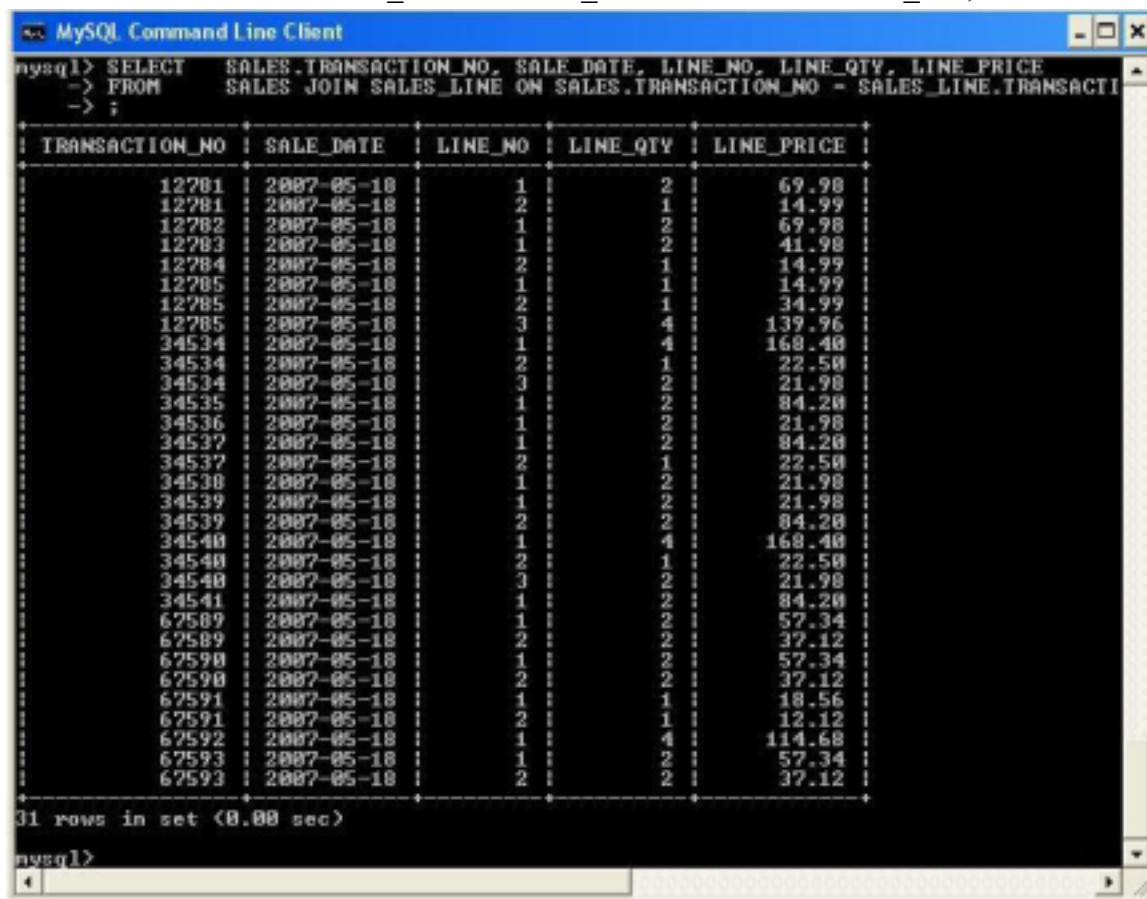
The join condition will typically include an equality comparison expression of two columns.

(The columns may or may not share the same name but, obviously, must have comparable data types.) The syntax is:

```
SELECT column-list FROM table1 JOIN table2 ON join-condition
```

The following example performs a join of the SALES and SALES_LINE tables, using the ON clause. The result is shown in Figure 52.

```
SELECT SALES.TRANSACTION_NO, SALE_DATE, LINE_NO, LINE_QTY, LINE_PRICE  
FROM SALES JOIN SALES_LINE  
ON SALES.TRANSACTION_NO = SALES_LINE.TRANSACTION_NO;
```



The screenshot shows a MySQL Command Line Client window with the following query and results:

```
mysql> SELECT SALES.TRANSACTION_NO, SALE_DATE, LINE_NO, LINE_QTY, LINE_PRICE  
-> FROM SALES JOIN SALES_LINE ON SALES.TRANSACTION_NO = SALES_LINE.TRANSACTION_NO  
-> ;
```

TRANSACTION_NO	SALE_DATE	LINE_NO	LINE_QTY	LINE_PRICE
12701	2007-05-18	1	2	69.98
12701	2007-05-18	2	1	14.99
12702	2007-05-18	1	2	69.98
12703	2007-05-18	1	2	41.98
12704	2007-05-18	2	1	14.99
12705	2007-05-18	1	1	14.99
12705	2007-05-18	2	1	34.99
12705	2007-05-18	3	4	139.96
34534	2007-05-18	1	4	168.40
34534	2007-05-18	2	1	22.50
34534	2007-05-18	3	2	21.98
34535	2007-05-18	1	2	84.20
34536	2007-05-18	1	2	21.98
34537	2007-05-18	1	2	84.20
34537	2007-05-18	2	1	22.50
34538	2007-05-18	1	2	21.98
34539	2007-05-18	1	2	21.98
34539	2007-05-18	2	2	84.20
34540	2007-05-18	1	4	168.40
34540	2007-05-18	2	1	22.50
34540	2007-05-18	3	2	21.98
34541	2007-05-18	1	2	84.20
67509	2007-05-18	1	2	57.34
67509	2007-05-18	2	2	37.12
67509	2007-05-18	1	2	57.34
67509	2007-05-18	2	2	37.12
67591	2007-05-18	1	1	18.56
67591	2007-05-18	2	1	12.12
67592	2007-05-18	1	4	114.68
67593	2007-05-18	1	2	57.34
67593	2007-05-18	2	2	37.12

31 rows in set (0.00 sec)

Figure 52: Query results for SALES JOIN SALES_LINE ON

Note that unlike the NATURAL JOIN and the JOIN USING operands, the JOIN ON clause

requires a table qualifier for the common attributes. If you do not specify the table qualifier, you will get a “column ambiguously defined” error message.

2.5. The Outer Join

An outer join returns not only the rows matching the join condition (that is, rows with matching values in the common columns), but also the rows with unmatched values. The ANSI standard defines three types of outer joins: left, right, and full. The left and right designations reflect the order in which the tables are processed by the DBMS. Remember that join operations take place two tables at a time. The first table named in the FROM clause will be the left side, and the second table named will be the right side. If three or more tables are being joined, the result of joining the first two tables becomes the left side; the third table becomes the right side.

LEFT OUTER JOIN

The left outer join returns not only the rows matching the join condition (that is, rows with matching values in the common column), but also the rows in the left side table with unmatched values in the right side table. The syntax is:

SELECT column-list

FROM *table1* LEFT [OUTER] JOIN *table2* ON *join-condition*

Page | 18

For example, the following query lists the park code, park name, and attraction name for all attractions and includes those Theme parks with no currently listed attractions:

SELECT THEMEPARK.PARK_CODE, PARK_NAME, ATTRACT_NAME FROM

THEMEPARK LEFT JOIN ATTRACTION ON

THEMEPARK.PARK_CODE = ATTRACTION.PARK_CODE;

The results of this query are shown in Figure 53.

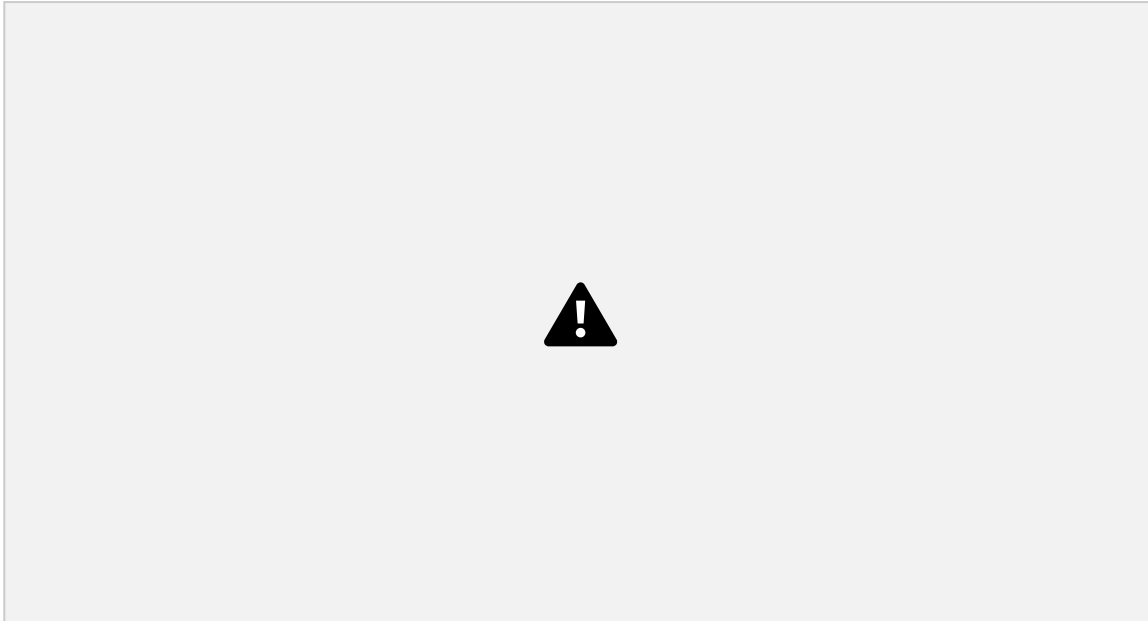


Figure 53: LEFT OUTER JOIN example

Task 6 Enter the query above and check your results with those shown in Figure 53.

RIGHT OUTER JOIN

The right outer join returns not only the rows matching the join condition (that is, rows with matching values in the common column), but also the rows in the right side table with unmatched values in the left side table. The syntax is:

SELECT column-list

FROM *table1* RIGHT [OUTER] JOIN *table2* ON *join-condition*

For example, the following query lists the park code, park name, and attraction name for all attractions and also includes those attractions that do not have a matching park code:

```
SELECT THEMEPARK.PARK_CODE, PARK_NAME, ATTRACT_NAME  
FROM THEMEPARK RIGHT JOIN ATTRACTION ON  
THEMEPARK.PARK_CODE = ATTRACTION.PARK_CODE;
```

The results of this query are shown in Figure 54.

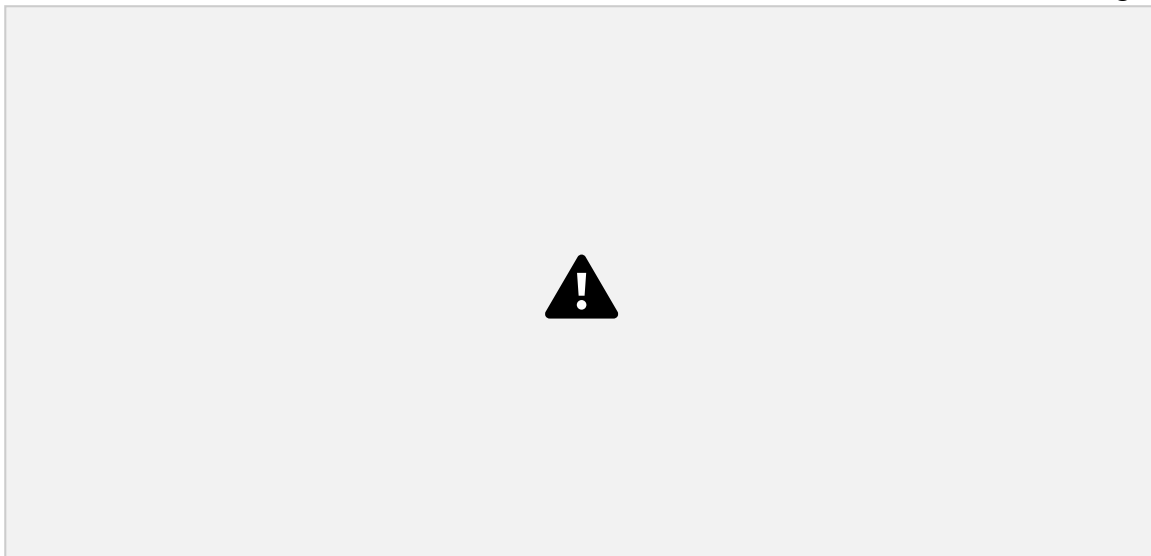


Figure 54: RIGHT OUTER JOIN example

Task 7 Enter the query above and check your results with those shown in Figure 54.

2.6. Cross Join

A **cross join** performs a relational product (also known as the Cartesian product) of two tables. The cross join syntax is:

```
SELECT column-list FROM table1 CROSS JOIN table2
```

For example,

```
SELECT * FROM SALES CROSS JOIN SALES_LINE;
```

performs a cross join of the SALES and SALES_LINE tables. That CROSS JOIN query generates 589 rows. (There were 19 sales rows and 31 SALES_LINE rows, thus giving $19 \times 31 = 589$ rows.)

Exercises

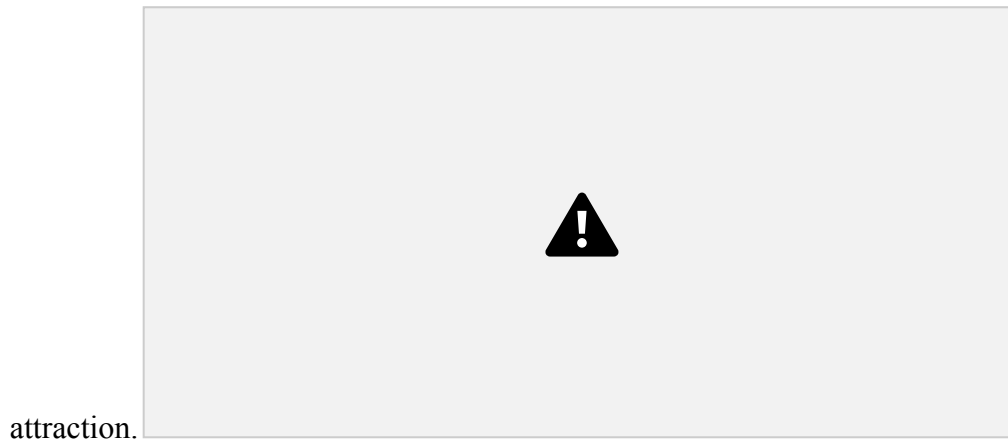
E4.1 Write a query that displays the average hourly rate that has been paid to all employees. Your query should return €7.03.

E4.2 Write a query that displays the average attraction age for all attractions where the PARK_CODE is equal to 'UK3452'. Your query should return 7.25 years.

E4.3 Display the employee numbers of all employees and the total number of hours they have worked. Output format

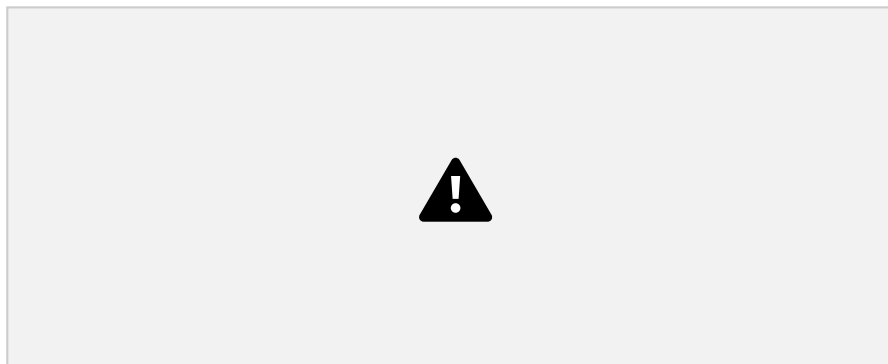


E4.4 Show the attraction number and the minimum and maximum hourly rate for each



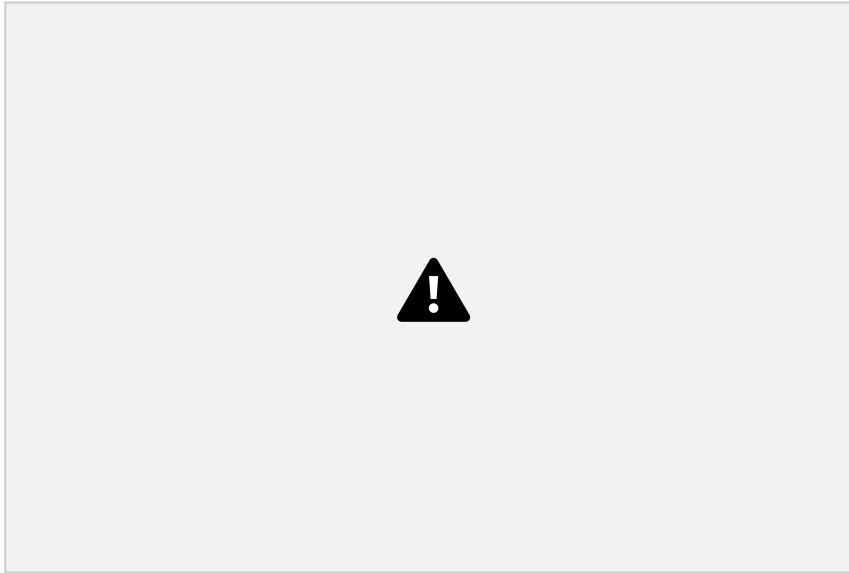
Page | 22

E4.5 Using the HOURS table, write a query to display the employee number (EMP_NUM), the attraction number (ATTRACT-NO) and the average hours worked per attraction (HOURS_PER_ATTRACT) limiting the result to where the average hours worked per attraction is greater or equal to 5. Check your results against those shown in below Figure.



E 4.6 Write a CROSS JOIN query which selects all rows from the EMPLOYEE and HOURS tables. How many rows were returned?

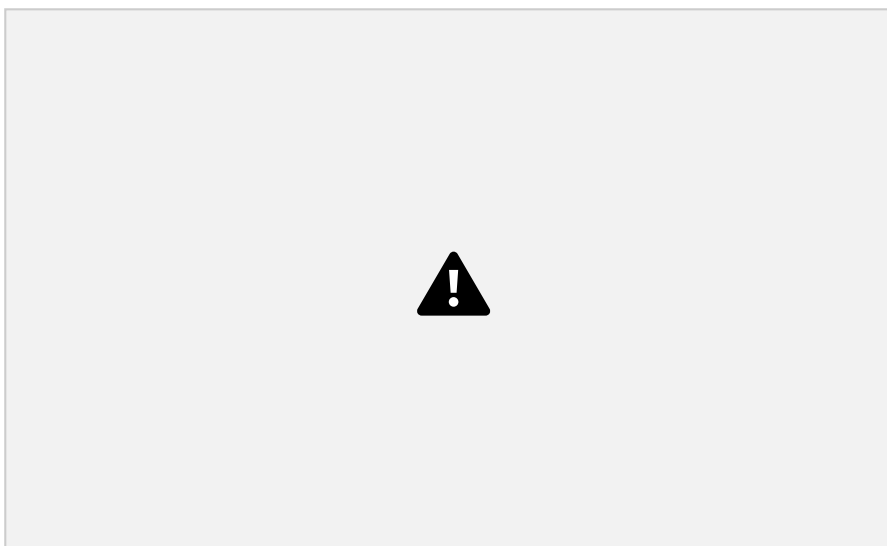
E.4.7 Write a query that displays the employees first and last name (EMP_FNAME and EMP_LNAME), the attraction number(ATTRACT_NO) and the date worked. ***Hint:*** (You will have to join the HOURS and the EMPLOYEE tables. Check your results with those shown in below figure.)



Page | 23

E4.8 Rewrite the query you wrote in **E 4.7** so that the attraction name (ATTRACT_NAME located in the ATTRACTION table) is also displayed.

Hint: (You will need to join three tables. Your output should match that shown in below Figure.)



E4.9 Display the park names and total sales for Theme Parks who are located in the country 'UK' or 'FR'.

E4.10 List the sale date, line quantity and line price of all transactions on the 18th May 2007.

Hint: (Remember the format of MySQL dates is '2007-05-18').