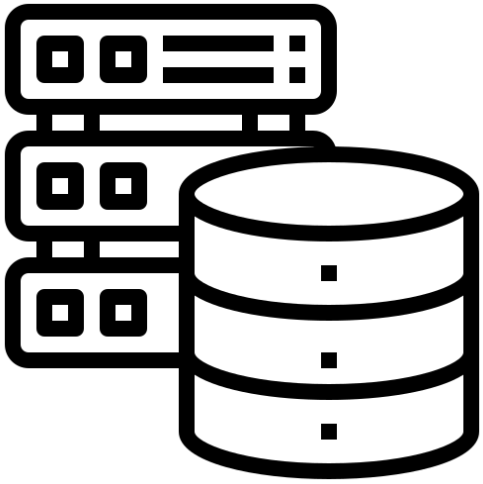
Name..... Nickname..... ID.....

05016226
Database System Concepts

Lab Tasks and Exercises



Dr.Nattaporn Chuenjarern

Dr.Jiraphat Yokrattanasak

Department of Mathematics, School of Science

King Mongkut's Institute of Technology Ladkrabang

Table of Contents

Introduction 1
Lab 1: Starting MySQL 2
Lab 2: Building a database: Table by Table 3
Lab 3: Data Manipulation Commands 8
Lab 4: Basic SELECT statements 15
Lab 5: Advanced SELECT Statements 23
Lab 6: Joining Database Tables 30
Lab 7: SQL Functions 41
Lab 8: Subqueries 55
Lab 9: Views 61

Introduction

There are 9 weeks lab. On completion of this 9 labs you will be able to:

- Create a simple relational database in MySQL.
- Insert, update and delete data the tables.
- Create queries using basic and advanced SELECT statements
- Perform join operations on relational tables
- Use aggregate functions in SQL
- Write subqueries
- Create views of the database

Tentative Schedule

Date	Lab
Friday 13 January 2023	**Lab 1** Starting MySQL
Friday 20 January 2023	**Lab 2** Building a database: Table by Table
Friday 27 January 2023	**Lab 3** Data Manipulation Commands
Friday 3 February 2023	**Lab 4** Basic SELECT statements
Friday 10 February 2023	**Lab 5** Advance SELECT statements
Friday 17 February 2023	Lab Midterm Exam
TBD	Demo Access/PHP
Friday 17 March 2023	**Lab 6** JOINING Database TABLES
Friday 24 March 2023	**Lab 7** SQL Functions
Friday 31 March 2023	**Lab 8** Subqueries
Friday 7 April 2023	**Lab 9** Views
Friday 21 April 2023	Lab Final Exam

Grading

Activity	Portion
Participants	5%
Assignments	5%
Lab Midterm Exam	5%
Lab Final Exam	5%

Lab 1: Starting MySQL

The learning objectives of this lab are to

- Learn how to start MySQL
- Learn how to use the MySQL command line client window
- Obtain help in MySQL

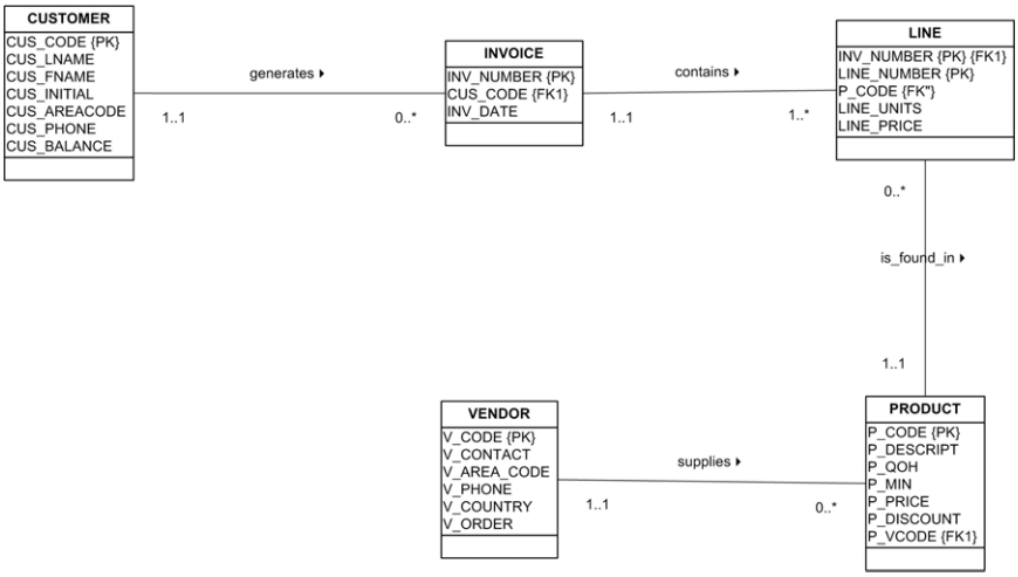


Figure The SaleCo Database ERD

Task: 1.1 Enter the following SQL command and write what the program shows

1. SHOW DATABASES;

2. USE SALECO;

3. SHOW TABLES;

Lab 2: Building a database: Table by Table

The learning objectives of this lab are to

- Create table structures using MySQL data types
- Apply SQL constraints to MySQL tables
- Create a simple index

2.1 The Theme Park Database

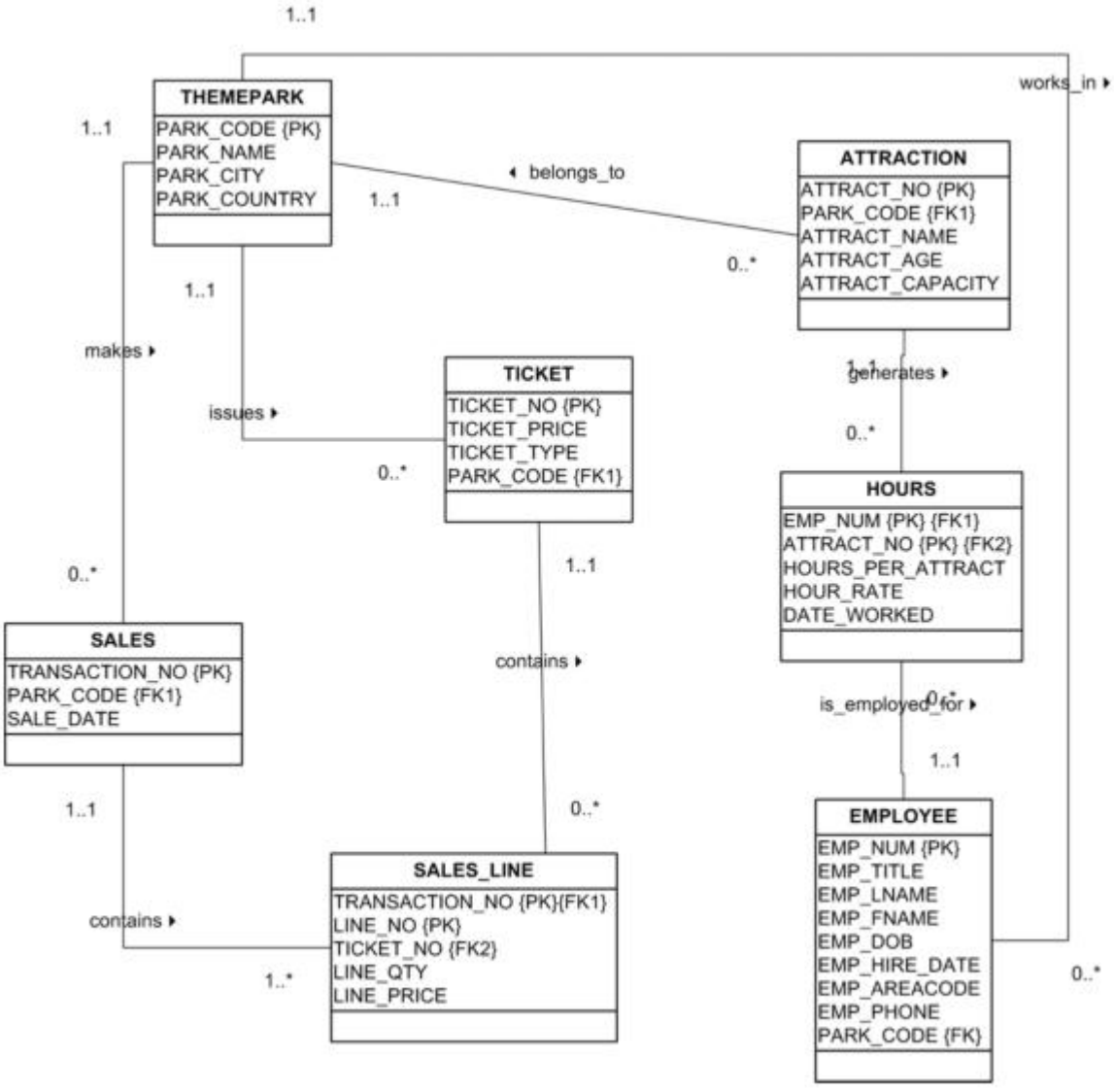


Figure 11 The Theme park Database ERD

2.2 Data Types in MySQL

The following are the example of data type in MySQL

Data Type	Example	Description
CHAR(size)	fieldName CHAR(10)	Stores up to 255 characters. If the content is smaller than the field size, the content will have trailing spaces appended.
VARCHAR(size)	fieldName VARCHAR(100)	Stores up to 255 characters, and a minimum of 4 characters. No trailing spaces are appended to the end of this datatype.
INT	fieldName INT	Stores a signed or unsigned integer number. Unsigned integers have a range of 0 to 4,294,967,295, and signed integers have a range of -2,147,438,648 to 2,147,438,647. By default, the INT data type is signed. To create an unsigned integer, use the UNSIGNED attribute.
FLOAT	fieldName FLOAT	Used for single precision floating point numbers.
DOUBLE	fieldName DOUBLE	Used for double precision floating point number
DATE	fieldName DATE	Stores dates in the format YYYY-MM-DD.
	fieldName TIME	TIME Stores times in the format HH:MM:SS.

2.3 Creating the Table Structures

- NOT NULL
- UNIQUE
- Primary Key - both a NOT NULL and a UNIQUE
- DEFAULT
- FOREIGN KEY

Task: 2.3.1 Creating the THEMEPARK Database.

- ✓ CREATE DATABASE THEMEMPARK;
- ✓ SHOW DATABASES;
- ✓ USE THEMEPARK;
- ✓ SHOW TABLES;

Task: 2.3.2 Creating the THEMEPARK TABLE

- ✓ CREATE TABLE THEMEPARK (
  PARK_CODE VARCHAR(10) PRIMARY KEY,
  PARK_NAME VARCHAR(35) NOT NULL,
  PARK_CITY VARCHAR(50) NOT NULL,
  PARK_COUNTRY CHAR(2) NOT NULL);

Task 2.3.3 Creating the EMPLOYEE TABLE

- ✓ CREATE TABLE EMPLOYEE (
  EMP_NUM NUMERIC(4) PRIMARY KEY,
  EMP_TITLE VARCHAR(4),
  EMP_LNAME VARCHAR(15) NOT NULL,
  EMP_FNAME VARCHAR(15) NOT NULL,
  EMP_DOB DATE NOT NULL,
  EMP_HIRE_DATE DATE,
  EMP_AREA_CODE VARCHAR(4) NOT NULL,
  EMP_PHONE VARCHAR(12) NOT NULL,
  PARK_CODE VARCHAR(10),
  CONSTRAINT FK_EMP_PARK FOREIGN KEY(PARK_CODE) REFERENCES
  THEMEPARK(PARK_CODE));

Task 2.3.4 Creating the TICKET TABLE

- ✓ CREATE TABLE TICKET (
  TICKET_NO NUMERIC(10) PRIMARY KEY,
  TICKET_PRICE NUMERIC(4,2) DEFAULT 00.00 NOT NULL,
  TICKET_TYPE VARCHAR(10),
  PARK_CODE VARCHAR(10),
  CONSTRAINT FK_TICKET_PARK FOREIGN KEY(PARK_CODE)
  REFERENCES THEMEPARK(PARK_CODE));

Task 2.3.5 Creating the ATTRACTION TABLE

- ✓ CREATE TABLE ATTRACTION (
  ATTRACT_NO NUMERIC(10) PRIMARY KEY,
  ATTRACT_NAME VARCHAR(35),
  ATTRACT_AGE NUMERIC(3) DEFAULT 0 NOT NULL,
  ATTRACT_CAPACITY NUMERIC(3) NOT NULL,
  PARK_CODE VARCHAR(10),
  CONSTRAINT FK_ATTRACT_PARK FOREIGN
  KEY(PARK_CODE) REFERENCES THEMEPARK(PARK_CODE));

Task 2.3.6 Creating the HOURS TABLE

- ✓ CREATE TABLE HOURS (
  EMP_NUM NUMERIC(4),
  ATTRACT_NO NUMERIC(10),
  HOURS_PER_ATTRACT NUMERIC(2) NOT NULL,

```
HOUR_RATE NUMERIC(4,2) NOT NULL,
DATE_WORKED DATE NOT NULL,
CONSTRAINT PK_HOURS PRIMARY KEY(EMP_NUM,
ATTRACT_NO,DATE_WORKED),
CONSTRAINT FK_HOURS_EMP FOREIGN KEY (EMP_NUM) REFERENCES
EMPLOYEE(EMP_NUM),
CONSTRAINT FK_HOURS_ATTRACT FOREIGN KEY
(ATTRACT_NO) REFERENCES ATTRACTION(ATTRACT_NO));
```

Task 2.3.7 Creating the SALES TABLE

- ✓
```
CREATE TABLE SALES (
TRANSACTION_NO NUMERIC PRIMARY KEY,
PARK_CODE VARCHAR(10),
SALE_DATE DATE NOT NULL,
CONSTRAINT FK_SALES_PARK FOREIGN KEY(PARK_CODE)
REFERENCES THEMEPARK(PARK_CODE));
```

Task 2.3.8 Creating the SALES_LINE TABLE

- ✓
```
CREATE TABLE SALES_LINE (
TRANSACTION_NO NUMERIC,
LINE_NO NUMERIC(2,0) NOT NULL,
TICKET_NO NUMERIC(10) NOT NULL,
LINE_QTY NUMERIC(4) DEFAULT 0 NOT NULL,
LINE_PRICE NUMERIC(9,2) DEFAULT 0.00 NOT NULL,
CONSTRAINT PK_SALES_LINE PRIMARY KEY (TRANSACTION_NO,LINE_NO),
CONSTRAINT FK_SALES_LINE_SALES FOREIGN KEY (TRANSACTION_NO)
REFERENCES SALES(TRANSACTION_NO) ON DELETE CASCADE,
CONSTRAINT FK_SALES_LINE_TICKET FOREIGN KEY (TICKET_NO)
REFERENCES TICKET(TICKET_NO));
```

2.4 Listing all tables

Task 2.4 Use the SHOW TABLES command to list all tables that have been created within the THEMEPARK database.

2.5 Altering the table structure

- ✓
```
ALTER TABLE ATTRACTION
MODIFY ATTRACT_CAPACITY NUMERIC(4);
```

2.6 Display a table's structure

The command **DESCRIBE** is used to display the structure of an individual table. To see the structure of the table you would enter the command:

DESCRIBE *name of table*

Lab 3: Data Manipulation Commands

The learning objectives for this lab are

- To know how to insert, update and delete data from within a table
- To learn how to retrieve data from a table using the SELECT statement

3.1 Adding Table

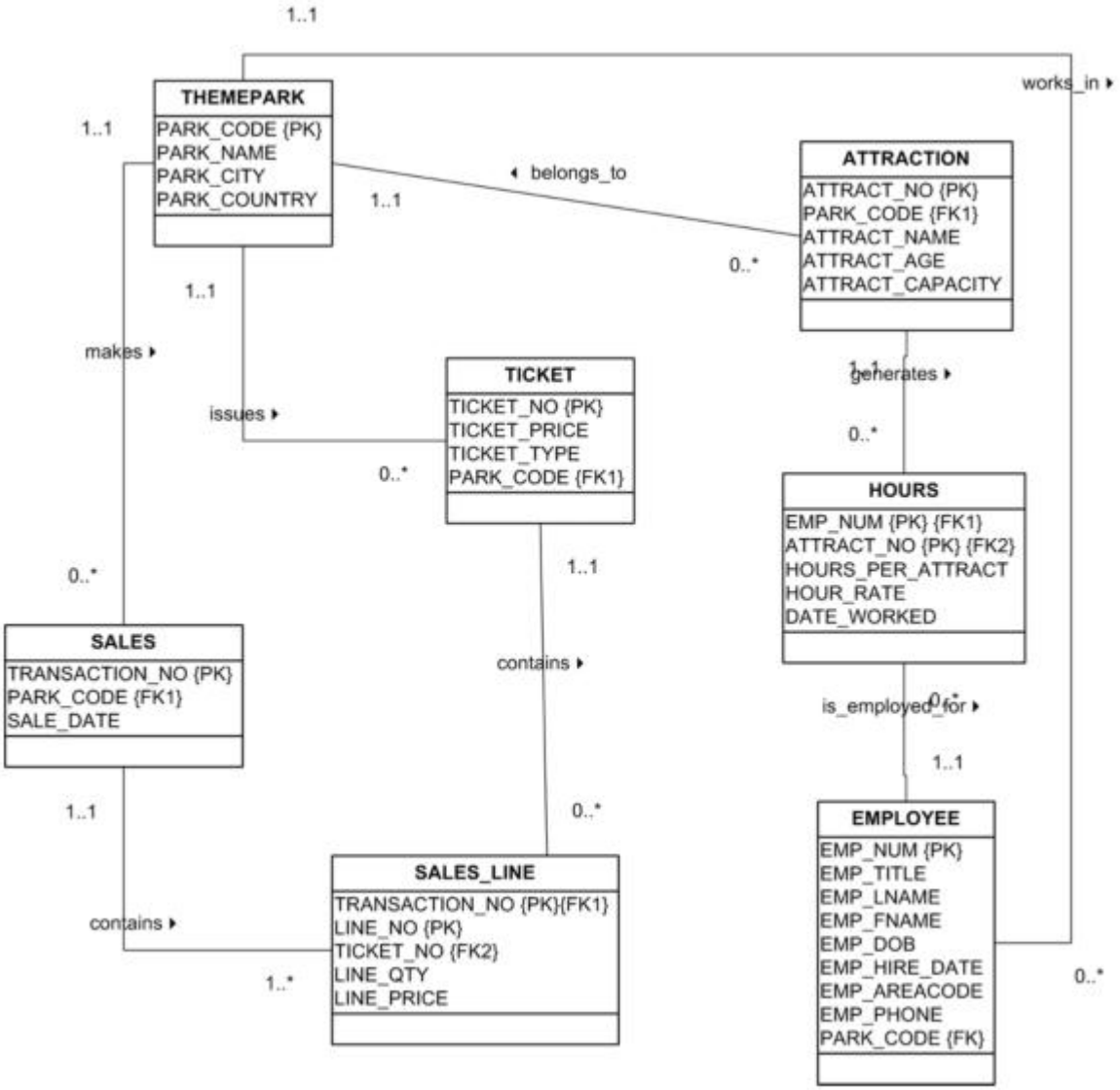


Figure 11 The Theme park Database ERD

SQL requires the use of the INSERT command to enter data into a table. The INSERT command's basic syntax looks like this:

INSERT INTO tablename VALUES (value1, value2, ..., valuen).

Task 3.1 Enter the first two rows of data into the THEMEPARK table using the following SQL insert commands.

- ✓ INSERT INTO THEMEPARK VALUES ('FR1001','FairyLand','PARIS','FR');
- ✓ INSERT INTO THEMEPARK VALUES ('UK3452','PleasureLand','STOKE','UK');

Task 3.2 Enter the following corresponding rows of data into the TICKET table using the following SQL insert commands.

- ✓ INSERT INTO TICKET VALUES (13001,18.99,'Child','FR1001');
- ✓ INSERT INTO TICKET VALUES (13002,34.99,'Adult','FR1001');
- ✓ INSERT INTO TICKET VALUES (13003,20.99,'Senior','FR1001');
- ✓ INSERT INTO TICKET VALUES (88567,22.50,'Child','UK3452');
- ✓ INSERT INTO TICKET VALUES (88568,42.10,'Adult','UK3452');
- ✓ INSERT INTO TICKET VALUES (89720,10.99,'Senior','UK3452');

Any changes made to the table contents are not physically saved on disk until you close the database, close the program you are using, or use the **COMMIT** command. The **COMMIT** command will permanently save any changes—such as rows added, attributes modified, and rows deleted—made to any table in the database.

Task 3.3 COMMIT the changes to the THEMEPARK and TICKET tables to the database.

Task 3.4 Run the script file *themeparkdata.sql* to insert the rest of the data into the Theme Park database.

You would enter the following command:

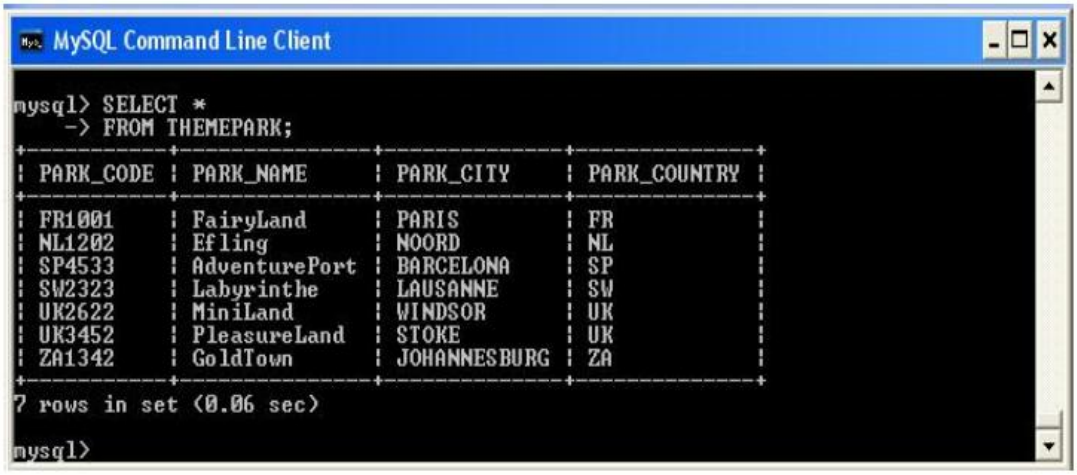
```
mysql> SOURCE  C:\path name\themeparkdata.sql
```

3.2 Retrieving data from a table using the SELECT Statement

The simplest query involves viewing all columns in one table. To display the details of all Theme Parks in the Theme Park database type the following:

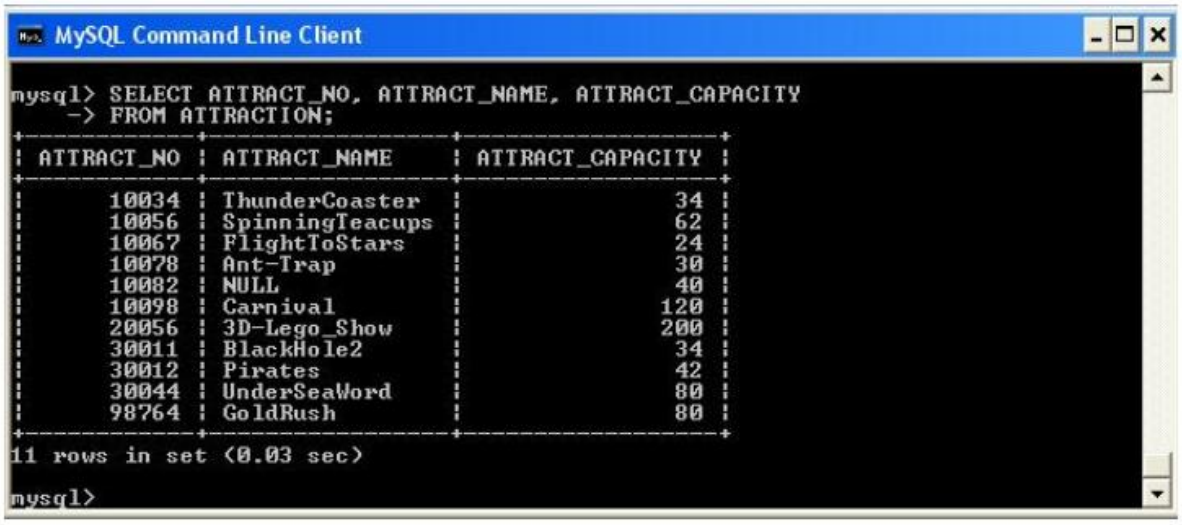
```
SELECT * FROM THEMEPARK;
```

You should see the output displayed in Figure

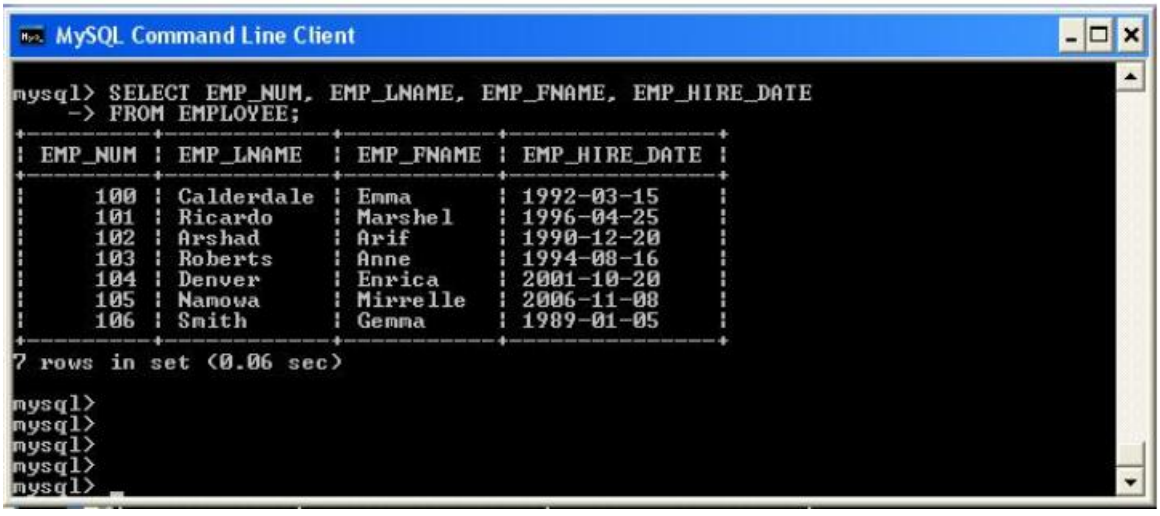


Task 3.5. Type in the following examples of the SELECT statement

1. SELECT ATTRACT_NO, ATTRACT_NAME, ATTRACT_CAPACITY FROM ATTRACTION;



2. SELECT EMP_NUM, EMP_LNAME, EMP_FNAME, EMP_HIRE_DATE FROM EMPLOYEE;



3.3 Updating table rows

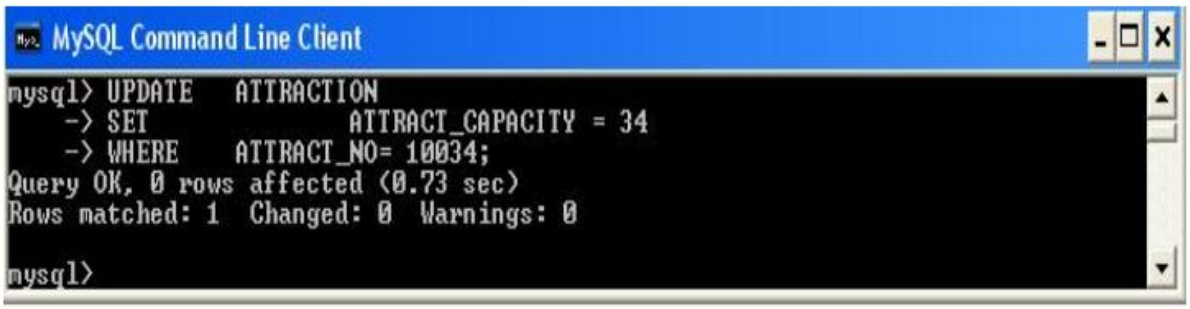
The UPDATE command is used to modify data in a table. The syntax for this command is:

```
UPDATE tablename
SET columnname = expression [, columnname = expression]
[WHERE conditionlist ];
```

For example, if you want to change the attraction capacity of the attraction number 10034 from 34, to 38. The primary key, ATTRACT_NO would be used to locate the correct (second) row, you would type:

```
UPDATE ATTRACTION
SET ATTRACT_CAPACITY = 34
WHERE ATTRACT_NO= 10034;
```

The output is shown in Figure.



Remember, the UPDATE command is a set-oriented operator. Therefore, if you don't specify a WHERE condition, the UPDATE command will apply the changes to all rows in the specified table.

Task 3.6 Enter the following SQL UPDATE command to update the age a person can go on a specific ride in a Theme Park.

```
UPDATE ATTRACTION
SET ATTRACT_AGE = 14;
```

Confirm the update by using this command to check the ATTRACTION table's listing:

```
SELECT * FROM ATTRACTION;
```

3.4 Restoring table contents

Task 3.7 To restore the data to their "pre-change" condition set the value, type the following commands;

```
SET AUTOCOMMIT = 0;
ROLLBACK;
```

Use the SELECT statement again to see that the ROLLBACK did, in fact, restore the data to their original values.

3.5 Deleting table rows

It is easy to delete a table row using the **DELETE** statement; the syntax is:

```
DELETE FROM tablename
[WHERE conditionlist ];
```

For example, if you want to delete a specific theme park from the THEMEPARK table
you could use the PARK_CODE as shown in the following SQL command:

```
DELETE FROM THEMEPARK
WHERE PARK_CODE = 'SW2323';
```

In that example, the primary key value lets SQL find the exact record to be deleted. However, deletions are not limited to a primary key match; any attribute may be used.

If you do not specify a WHERE condition, *all* rows from the specified table will be deleted!

3.6 Inserting Table rows with a subquery

Task 3.8 Use the following steps to populate your EMPLOYEE table.

- ✓ Run the script *emp_copy.sql* This script creates a table called EMP_COPY which we will populate using data from the EMPLOYEE table in the THEMEPARK database.
- ✓ Add the rows to EMP_COPY table by copying all rows from EMPLOYEE.

```
INSERT INTO EMP_COPY SELECT * FROM EMPLOYEE;
```

- ✓ Permanently save the changes: COMMIT;

Exercises Lab 3

Exercise 3.1 Load and run the script *park_copy.sql* which creates the PARK_COPY table.

Exercise 3.2 Describe the PARK_COPY and THEMEPARK tables and notice that they are different.

Exercise 3.3 Update the AREA_CODE and PARK_PHONEs fields in the PARK_COPY table with the following values.

PARK_CODE	PARK_AREA_CODE	PARK_PHONE
FR1001	5678	223-556
UK3452	0181	678-789
ZA1342	8789	797-121

Exercise 3.4 Add the following new theme parks to the PARK_COPY TABLE.

PARK_CODE	PARK_NAME	PARK_COUNTRY	PARK_AREA_CODE	PARK_PHONE
AU1001	SkiWorld	AU	1212	440-232
GR5001	RoboLand	GR	4565	123-123

Lab 4: 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

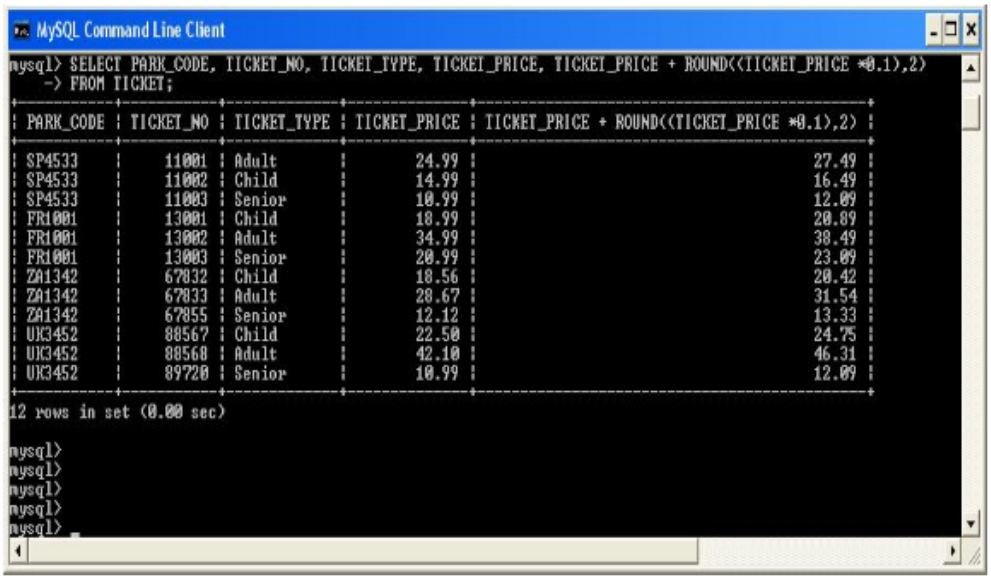
4.1 Using arithmetic operators in SQL statements

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 4.1 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;



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:

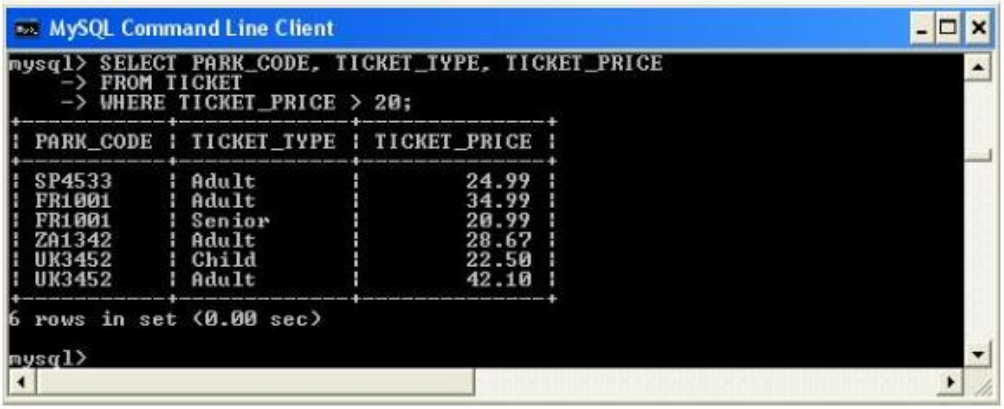
- ✓ SELECT PARK_CODE, TICKET_NO, TICKET_TYPE, TICKET_PRICE, TICKET_PRICE + ROUND((TICKET_PRICE *0.1),2) AS "PRICE INCREASE"
  FROM TICKET;

4.2 Selecting rows with conditional restrictions

Comparison

Task 4.2 Type in and execute the query and test out the greater than operator. Do you get the same results has shown in Figure?

- ✓ SELECT PARK_CODE, TICKET_TYPE, TICKET_PRICE
  FROM TICKET
  WHERE TICKET_PRICE > 20;

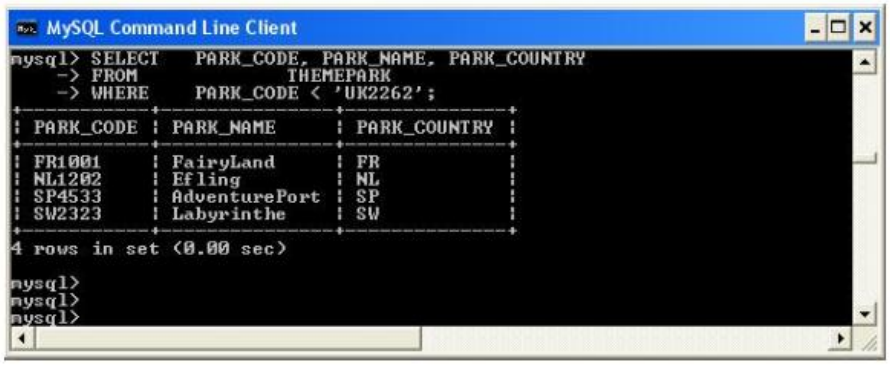


Task 4.3 Modify the query you have just executed to display tickets that are less than €30.00.

Character comparisons

Task 4.4 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

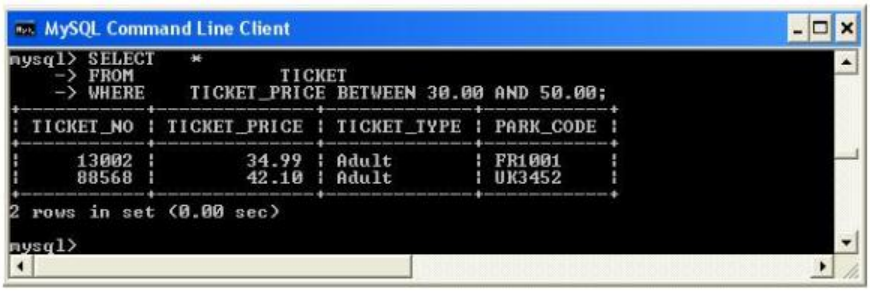
- ✓ SELECT PARK_CODE, PARK_NAME, PARK_COUNTRY
  FROM THEMEPARK
  WHERE PARK_CODE < 'UK2262';



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;

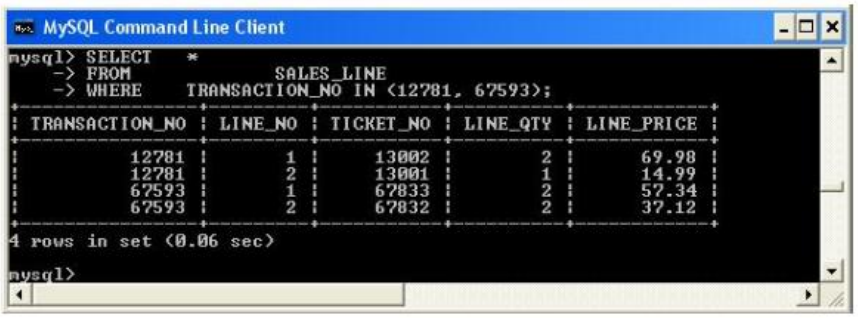


Task 4.5 Write a query which displays the employee number, attraction no, the hours worked per attraction and the date worked where the hours worked per attraction is between 5 and 10. Hint you will need to select data from the HOURS table.

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);



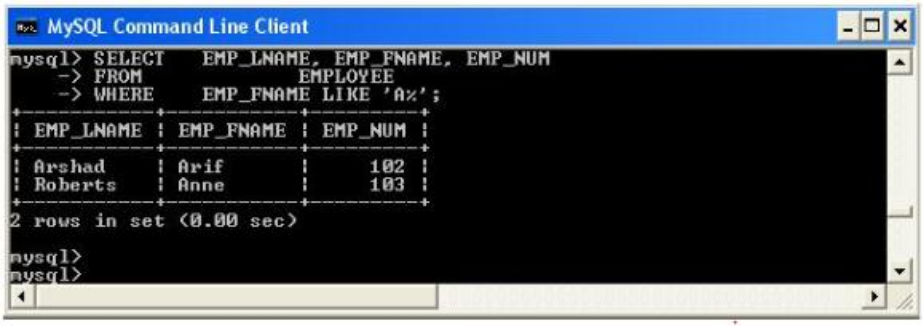
Task 4.6 Write a query to display all tickets that are of type Senior or Child.
Hint: Use the TICKET table.

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 4.7 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%';

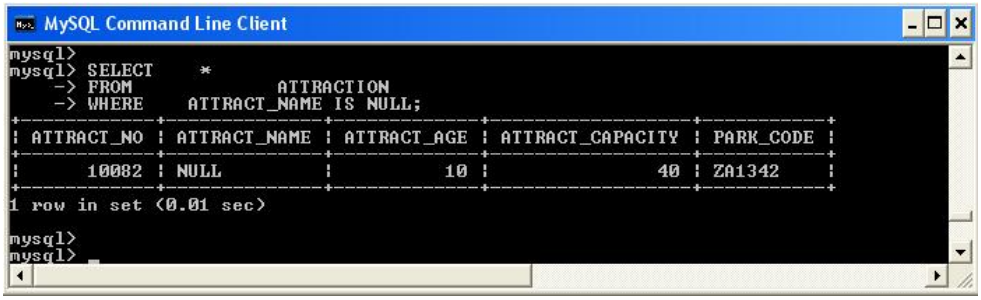


Task 4.8 Write a query which finds all Theme Parks that have a name ending in 'Land'.

NULL and IS NULL

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;



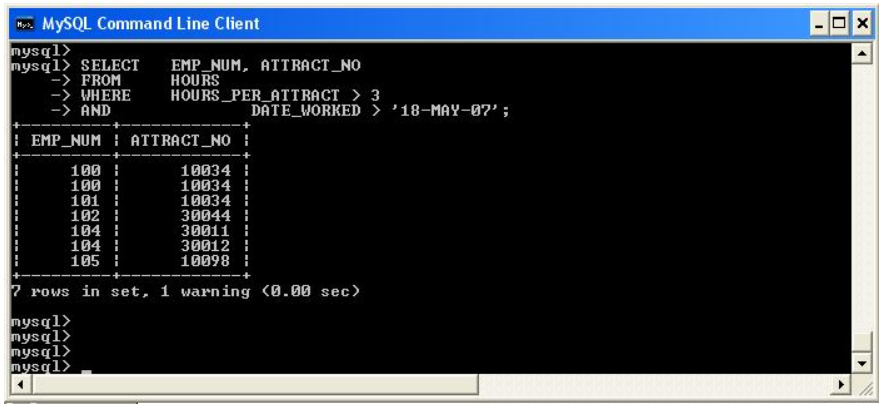
Logical Operators

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 > '18-MAY-07';

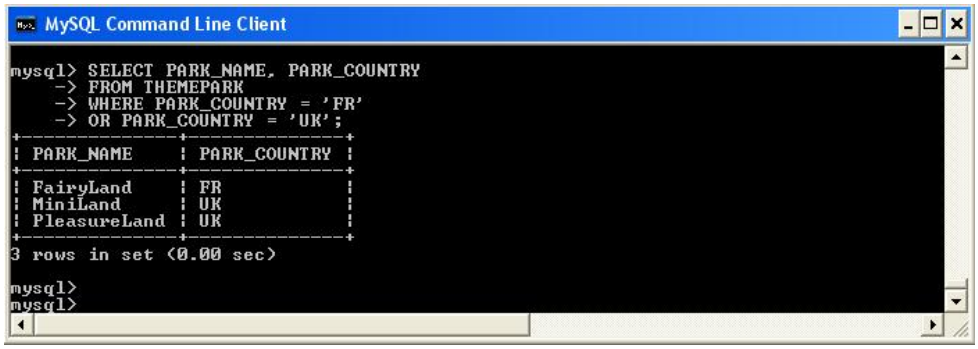


Task 4.9 Write a query which displays the details of all attractions which are suitable for children aged 10 or under and have a capacity of less than 100. You should not display any information for attractions which currently have no name.

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';



Task 4.11 Test the following query and explain out what this query is doing.

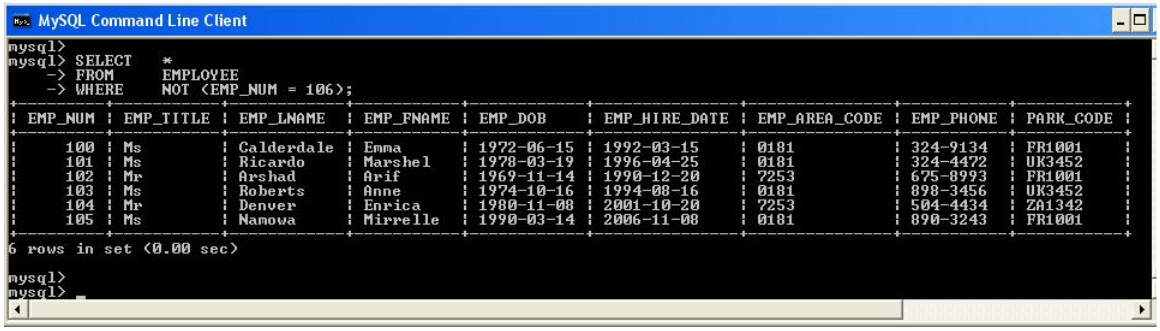
- ✓ SELECT *FROM ATTRACTION
  WHERE (PARK_CODE LIKE 'FR%'
  AND ATTRACT_CAPACITY <50) OR (ATTRACT_CAPACITY > 100);

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);



Exercises Lab 4

Exercise 4.1 Write a query to display all Theme Parks except those in the UK.

Exercise 4.2 Write a query to display all the sales that occurred on the 18th May 2007.

Exercise 4.3 Write a query to display the ticket prices between €20 AND €30.

Exercise 4.4 Display all attractions that have a capacity of more than 60 at the Theme Park FR1001.

Exercise 4.5 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.

Lab 5: Advanced SELECT Statements

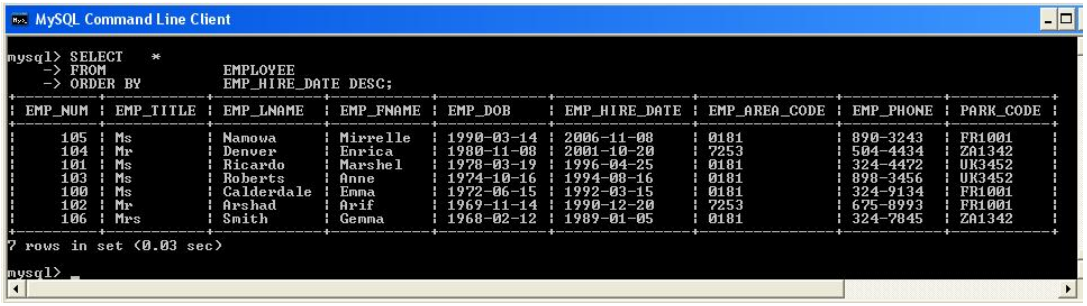
The learning objectives of this lab are to

- Sort the data in the resulting query
- Apply SQL aggregate functions

5.1 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.

✓ SELECT * FROM EMPLOYEE
ORDER BY EMP_HIRE_DATE DESC;



EMP_NUM	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_DOB	EMP_HIRE_DATE	EMP_AREA_CODE	EMP_PHONE	PARK_CODE
105	Ms	Namowa	Mirrelle	1990-03-14	2006-11-08	0181	890-3243	FR1001
104	Mr	Denver	Enrica	1980-11-08	2001-10-20	7253	504-4434	ZA1342
101	Ms	Ricardo	Marshel	1978-03-19	1996-04-25	0181	324-4472	UK3452
103	Ms	Roberts	Anne	1974-10-16	1994-08-16	0181	898-3456	UK3452
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
106	Mrs	Smith	Gemma	1968-02-12	1989-01-05	0181	324-7845	ZA1342

Task 5.1 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;

Task 5.2 Enter the following query and 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;

5.2 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;

5.3 Aggregate Functions

SQL can perform mathematical summaries through the use of aggregate functions (Count, Min, Max, Sum, Avg). Aggregate functions return results based on groups of rows. By default, the entire result is treated as one group.

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;

However, if you wanted to know how many different Theme parks were in the ATTRACTION table, you would modify the query as follows

- ✓ SELECT COUNT(DISTINCT(PARK_CODE)) FROM ATTRACTION;

Task 5.3 Write a query that displays the number of distinct employees in the HOURS table. You should label the column "Number of Employees".

Task 5.4 Enter the following two queries and examine their output. Can you explain why the number of rows returned is different?

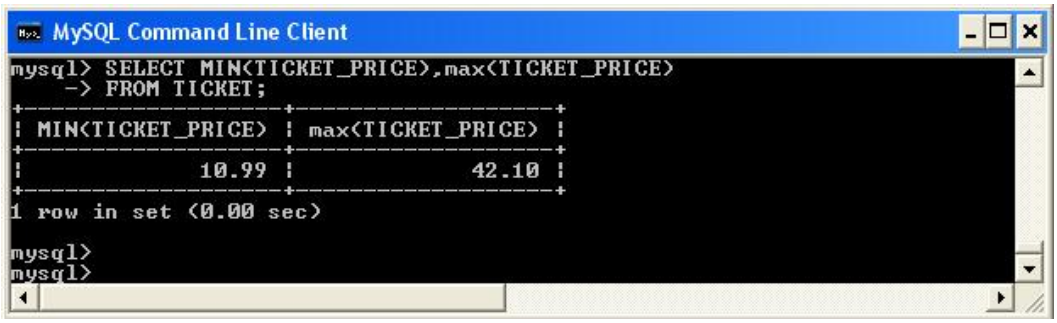
- ✓ SELECT COUNT(*) FROM ATTRACTION;
- ✓ SELECT COUNT(ATTRACT_NAME) FROM ATTRACTION;

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 5.5 Enter the following query which illustrates the use of the MIN and Max functions.

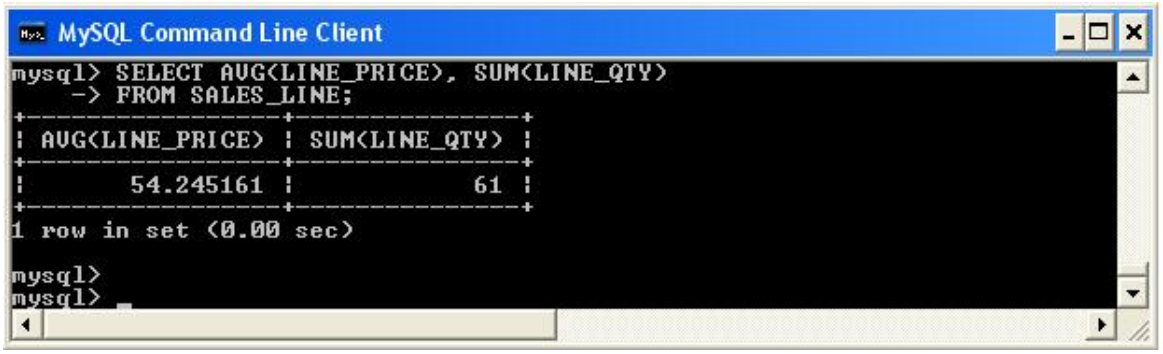
- ✓ SELECT MIN(TICKET_PRICE),max(TICKET_PRICE) FROM TICKET;



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).

- ✓ SELECT AVG(LINE_PRICE), SUM(LINE_QTY) FROM SALES_LINE;



Task 5.6 Write a query that displays the average hourly rate that has been paid to all employees. Hint use the HOURS table. Your query should return €7.03.

Task 5.7 Write a query that displays the average attraction age for all attractions where the PARK_CODE = 'UK3452'. Your query should return 7.25 years.

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 following query displays the minimum and maximum ticket price of all parks.

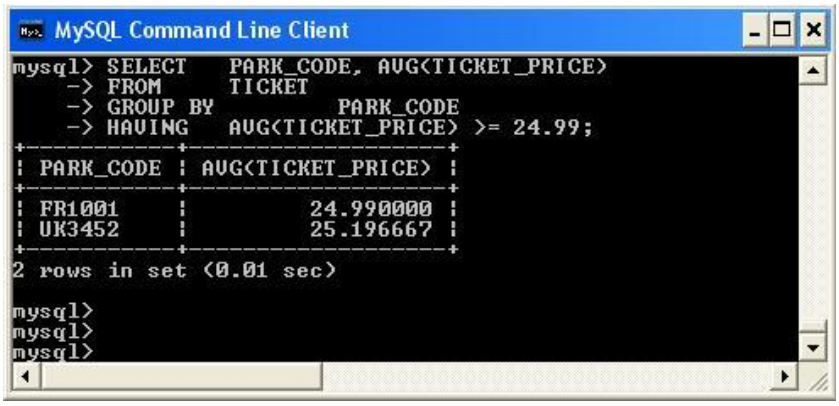
- ✓ SELECT PARK_CODE, MIN(TICKET_PRICE),MAX(TICKET_PRICE)
  FROM TICKET
  GROUP BY PARK_CODE;

Task 5.7 Enter the query above and check the results. 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. 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

- ✓ SELECT PARK_CODE, AVG(TICKET_PRICE) FROM TICKET
  GROUP BY PARK_CODE
  HAVING AVG(TICKET_PRICE) >= 24.99;



Task 5.8 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.

Exercises Lab 5

Exercise 5.1 Write a query to display all unique employees that exist in the HOURS table;

Exercise 5.2 Display the employee numbers of all employees and the total number of hours they have worked.

Exercise 5.3. Show the attraction number and the minimum and maximum hourly rate for each attraction.

Exercise 5.4 Write a query to show the transaction numbers and line prices (in the SALES_LINE table) that are greater than €50.

Exercise 5.5 Update data by the following

- ✓ UPDATE SALES
  SET SALE_DATE = DATE('2007-05-20')
  WHERE TRANSACTION_NO= '67592';
- ✓ UPDATE SALES
  SET SALE_DATE = DATE('2007-05-19')
  WHERE TRANSACTION_NO= '67593';

Write a query to display all information from the SALES table in descending order of the sale date.

Lab 6: Joining Database Tables

The learning objectives of this lab are to

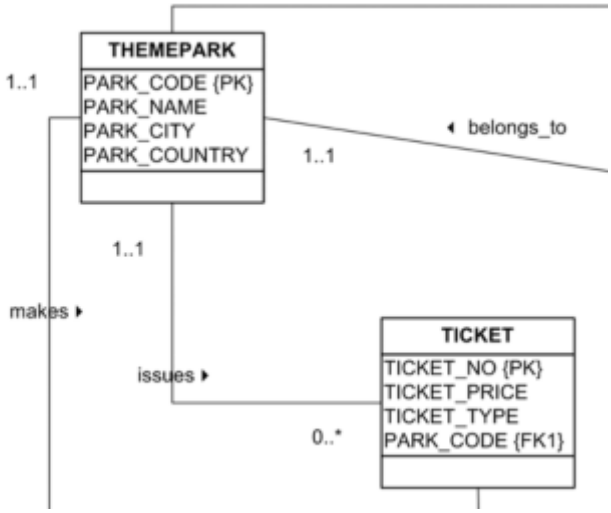
- Learn how to perform the following types of database joins
  - Cross Join
  - Natural Join
  - Outer Joins

6.1 Introduction to Joins

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

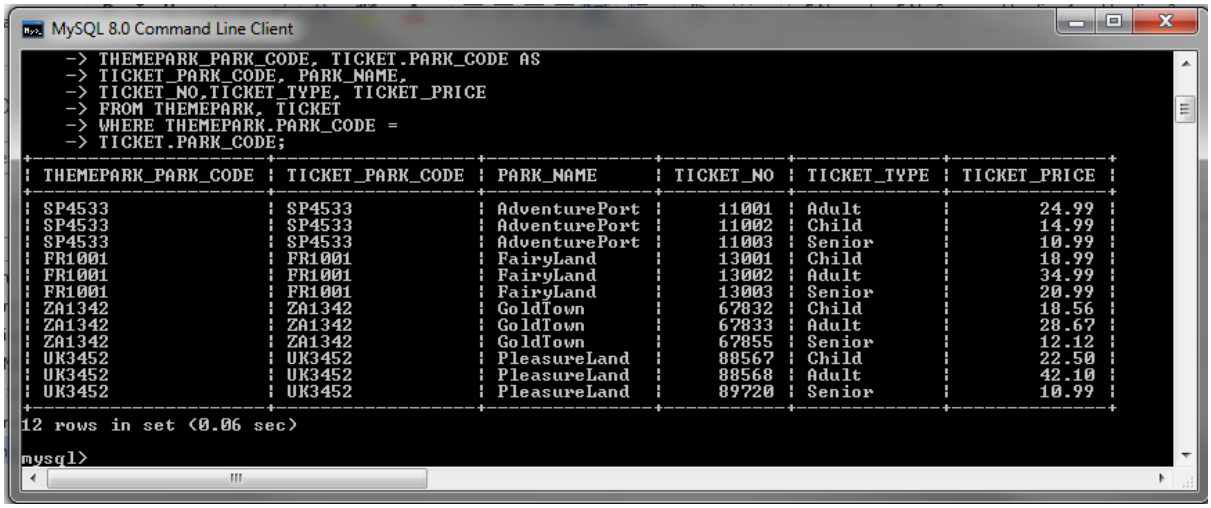
- Have common values in common columns (natural join)
- Meet a given join condition (equality or inequality)
- Have common values in common columns or have no matching values (outer join)

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 listed in the SELECT command sequence must be defined.



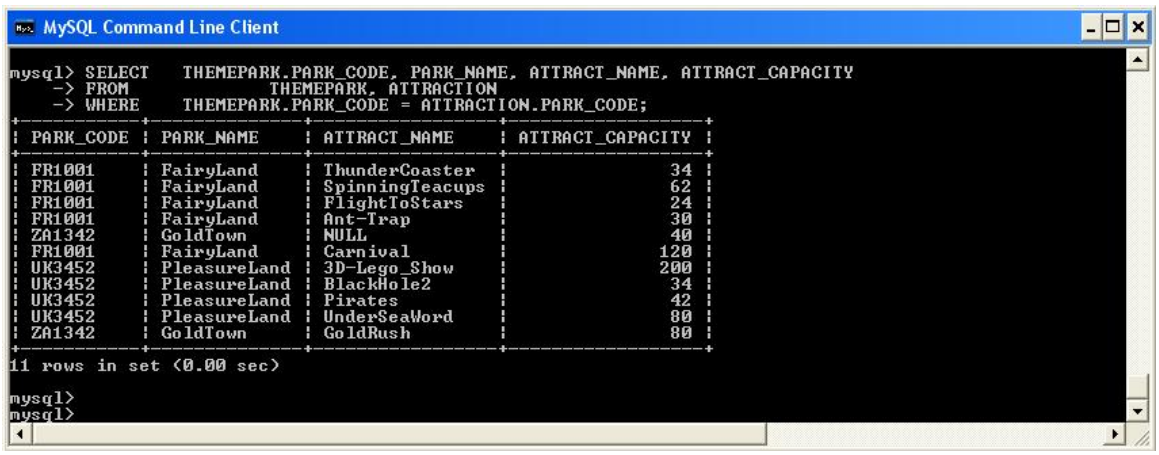
To join the THEMEPARK and TICKET tables, you would use the following,

- ✓ SELECT THEMEPARK.PARK_CODE AS THEMEPARK_PARK_CODE,
  TICKET.PARK_CODE AS TICKET_PARK_CODE, PARK_NAME,
  TICKET_NO, TICKET_TYPE, TICKET_PRICE
  FROM THEMEPARK, TICKET
  WHERE THEMEPARK.PARK_CODE = TICKET.PARK_CODE;



Task 6.1 Execute the following query and check your results with those shown in Figure

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



Then modify the SELECT statement to

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

What happens?

6.2 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 the before Task 6.1). Any legal table name may be used as an alias.

```
✓ SELECT P.PARK_CODE AS THEMEPARK_PARK_CODE,
        T.PARK_CODE AS TICKET_PARK_CODE, PARK_NAME,
        TICKET_NO, TICKET_TYPE, TICKET_PRICE
  FROM THEMEPARK P, TICKET T
  WHERE P.PARK_CODE =T.PARK_CODE;
```

6.3 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 X 31 = 589 rows.)

Task 6.2 Write a CROSS JOIN query which selects all rows from the EMPLOYEE and HOURS tables. How many rows were returned?

6.4 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:

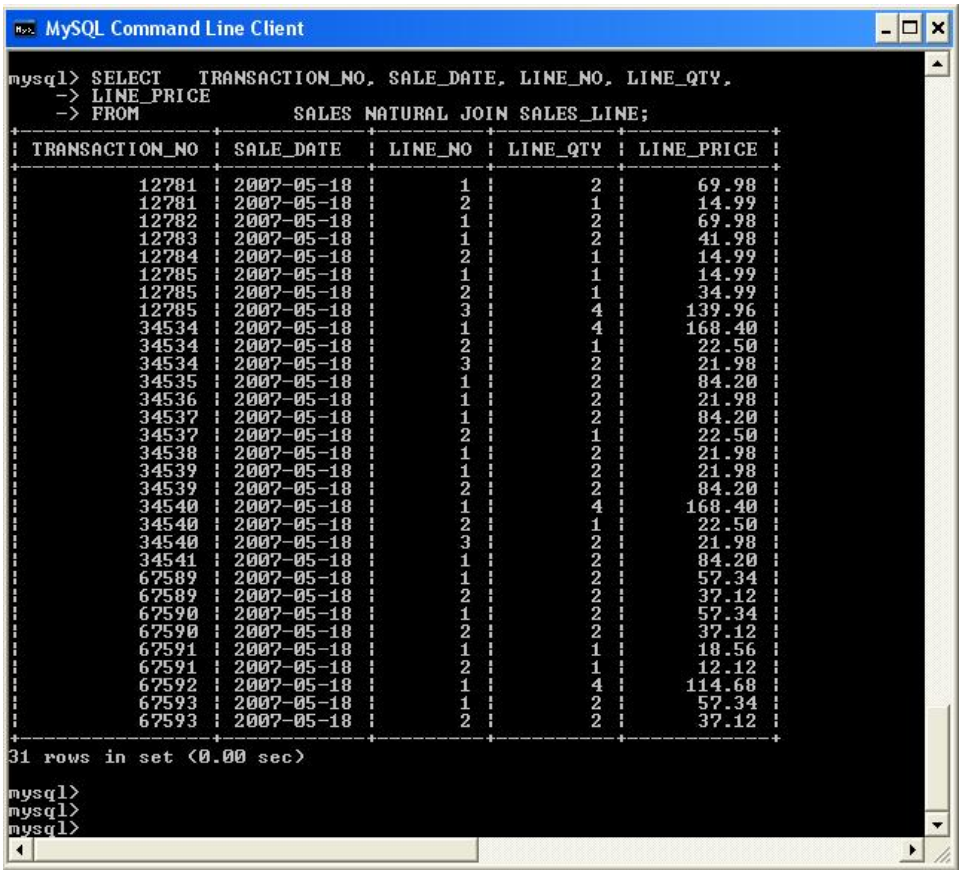
- ✓ SELECT column-list FROM table1 NATURAL JOIN table2

The **natural join** will perform the following tasks:

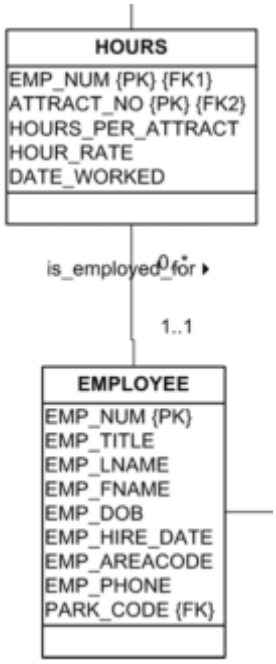
- Determine the common attribute(s) by looking for attributes with identical names and compatible data types
- Select only the rows with common values in the common attribute(s)
- If there are no common attributes, return the relational product of the two tables

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

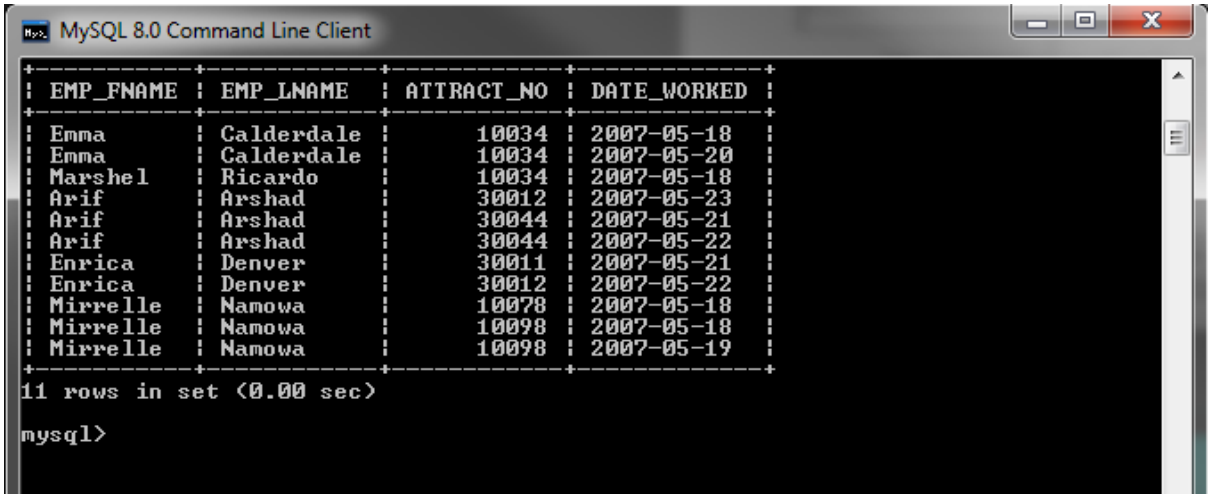
- ✓ SELECT TRANSACTION_NO, SALE_DATE, LINE_NO, LINE_QTY, LINE_PRICE FROM SALES NATURAL JOIN SALES_LINE;



Task 6.3 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 Figure.



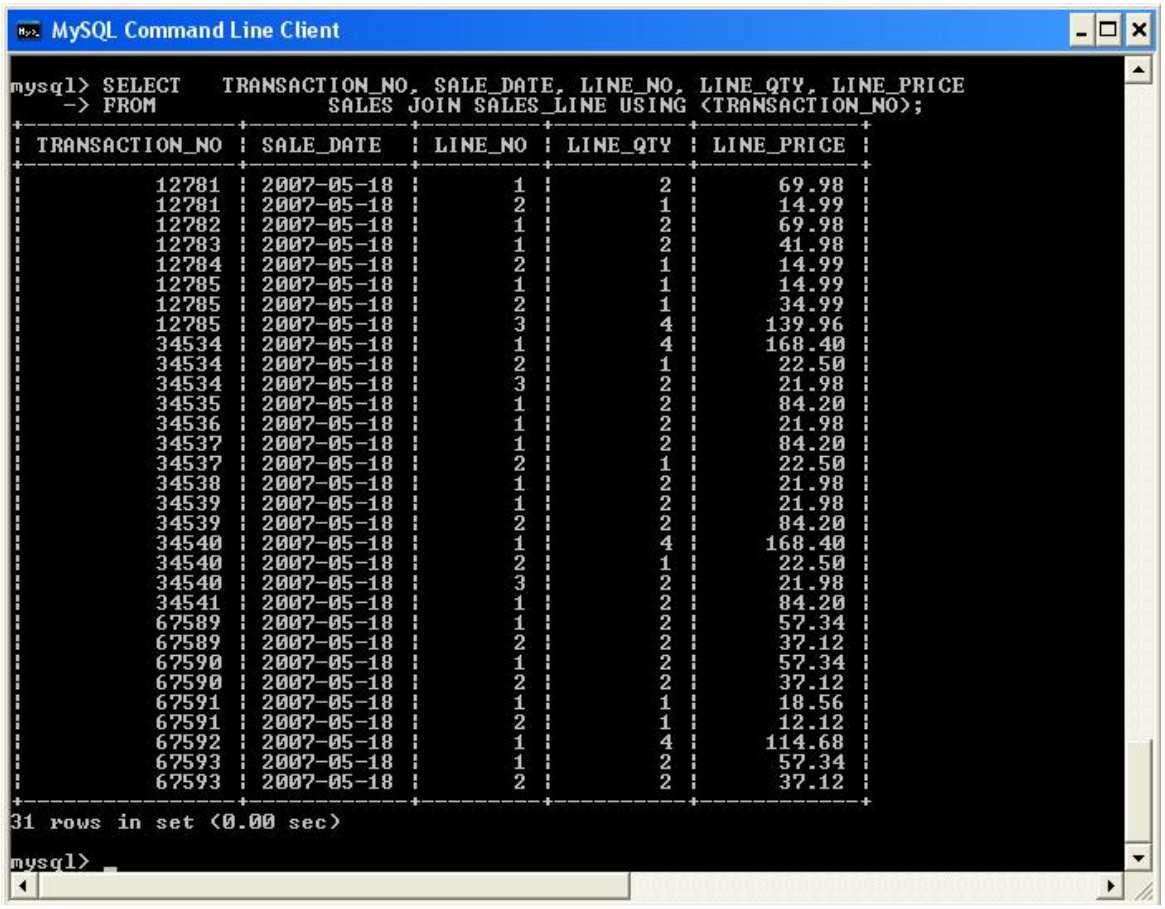
6.5 Join USING

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);



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

6.6 Join ON

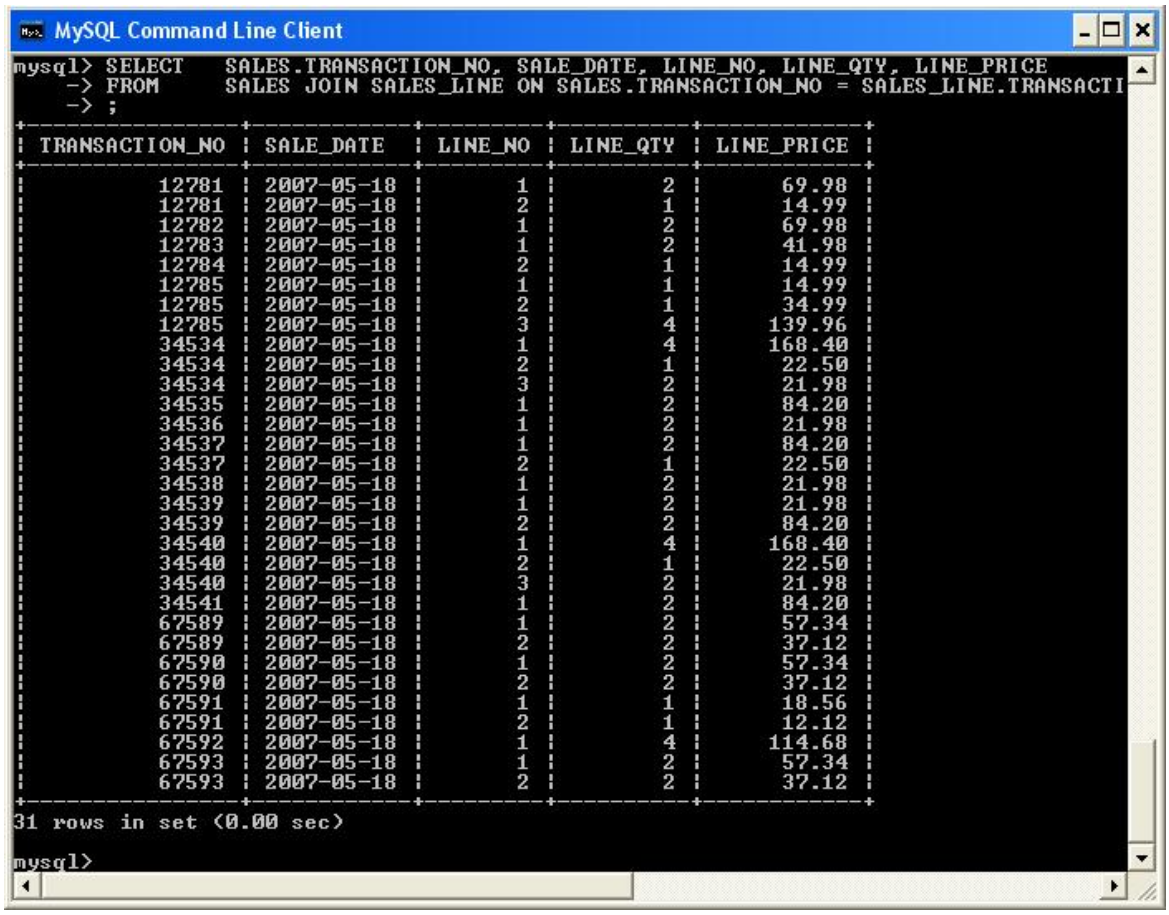
The previous two join styles used common attribute names in the joining tables. Another way to express a join when the tables have no common attribute names is to use the **JOIN ON** operand. That query will return only the rows that meet the indicated join condition. The join condition will typically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

- ✓ 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
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

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.

6.7 The Outer Join

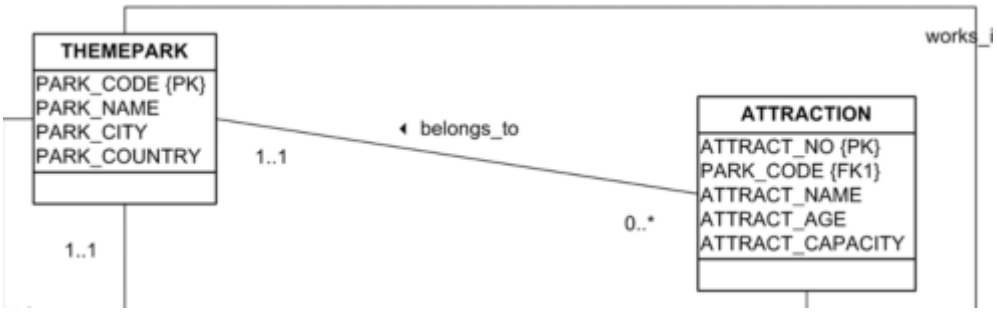
An outer join returns not only the rows matching the join condition (that is, rows with matching values in the common columns), but also the rows with unmatched values.

LEFT OUTER JOIN

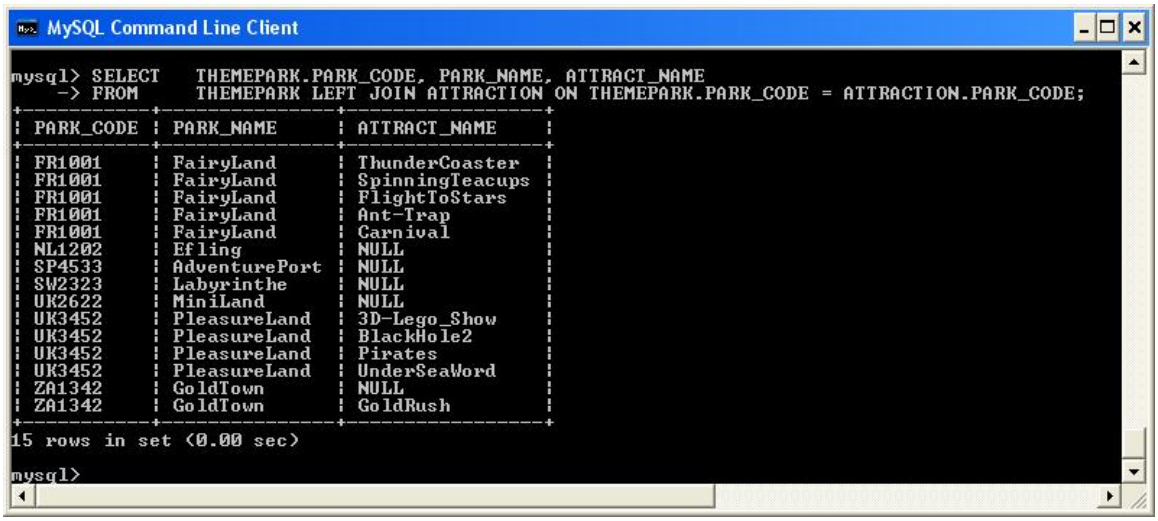
The left outer join returns not only the rows matching the join condition (that is, rows with matching values in the common column), but also the rows in the left side table with unmatched values in the right side table. The syntax is:

- ✓ SELECT column-list
  FROM table1 LEFT [OUTER] JOIN table2 ON join-condition

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;



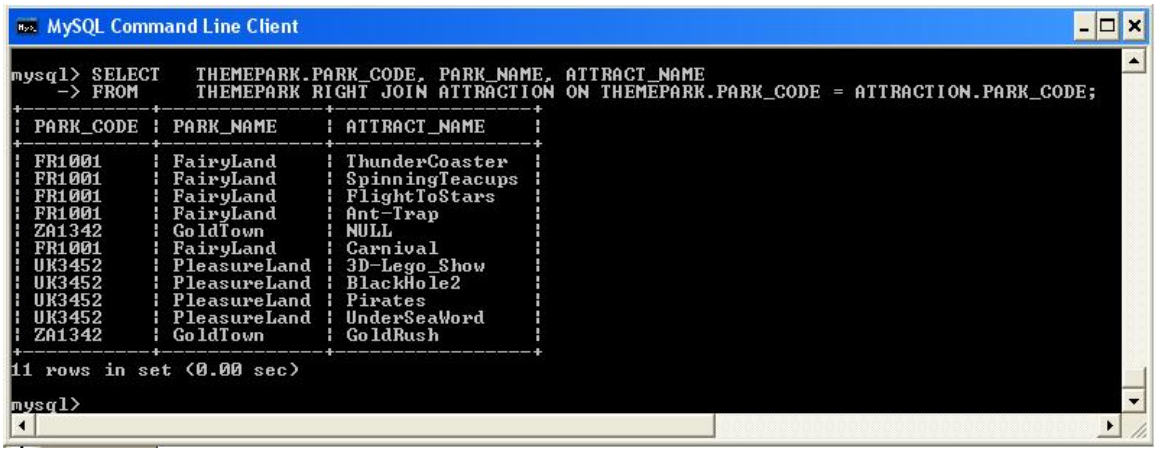
RIGHT OUTER JOIN

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;



Exercises Lab 6

Exercise 6.1 Use the cross join to display all rows in the EMPLOYEE and HOURS tables. How many rows were returned?

Exercise 6.2 Write a query to display the attraction number, employee first and last names and the date they worked on the attraction. Order the results by the date worked.

Exercise 6.3 Display the park names and total sales for Theme Parks who are located in the country 'UK' or 'FR'.

Exercise 6.4 Write a query to display the names of attractions that currently have not had any employees working on them.

Exercise 6.5 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').

Lab 7: SQL Functions

The learning objectives of this lab are to

- Learn about selected MySQL date and time functions
- Be able to perform string manipulations
- Utilize single row numeric functions
- Perform conversions between data types

7.1 Date and Time Functions

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

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

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

- ✓ SELECT DISTINCT(SALE_DATE) FROM SALES;

- ✓ SELECT DISTINCT(DATE_FORMAT(SALE_DATE, '%D %b %Y')) FROM SALES;

Table: taken directly from the MySQL Manual 5.0 shows a complete list of specifiers that can be used in the *format* string.

Specifier	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, ...)
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)
%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00..53), where Sunday is the first day of the week
%u	Week (00..53), where Monday is the first day of the week
%V	Week (01..53), where Sunday is the first day of the week; used with %X
%v	Week (01..53), where Monday is the first day of the week; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric (two digits)
%%	A literal '%' character
%x	*x*, for any '*x*' not listed above

Task 7.2 Using the date specifiers in Table, modify the query in **Task 7.1** #2 to display the date in the format 'Fri – 18 – 5 – 07'.

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

✓ SELECT CURRENT_DATE(), CURRENT_TIME();

MONTH, DAYOFMONTH and YEAR

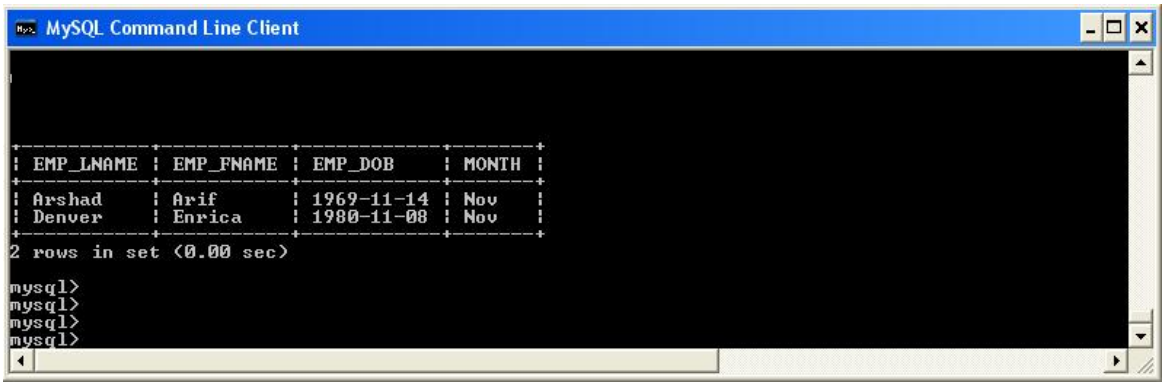
MySQL provides functions for extracting the month, day or year from any given date. The syntax of each function is as follows:

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.

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

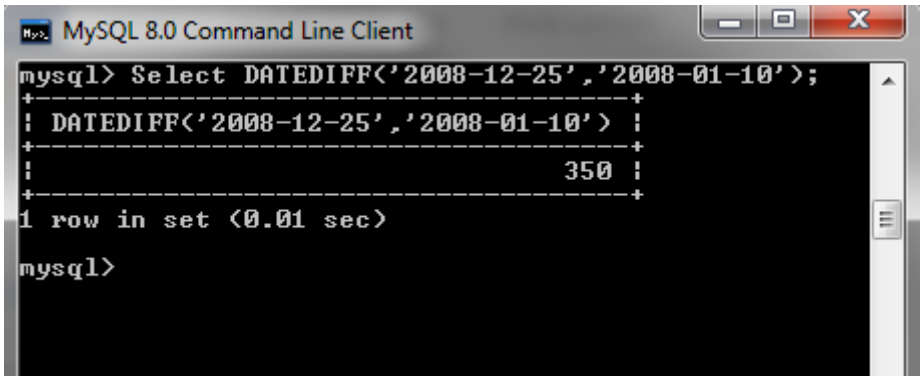
Task 7.3 Write a query that displays all employees who were born in November. Your output should match that shown in Figure.



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','2008-01-10');

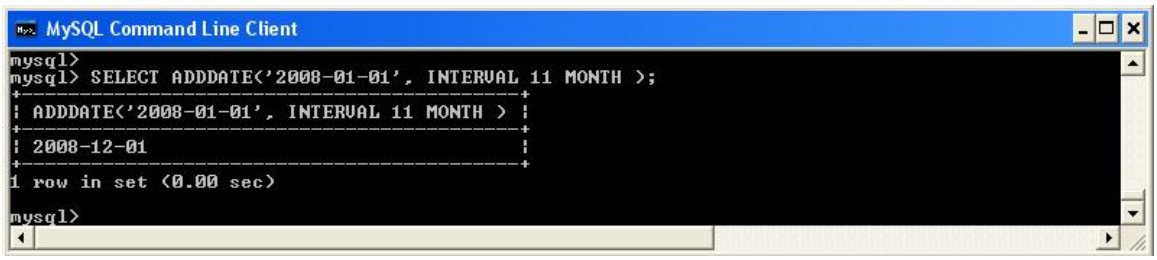


DATE_ADD and DATE_SUB

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)

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.



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

- ✓ SELECT EMP_LNAME, EMP_FNAME, EMP_HIRE_DATE, ADDDATE(EMP_HIRE_DATE, INTERVAL 12 MONTH) AS APPRAISAL FROM EMPLOYEE;

LAST_DAY

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

- ✓ LAST_DAY(date_value).

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

- ✓ SELECT * FROM SALES
  WHERE SALE_DATE >= LAST_DAY(SALE_DATE)-20;

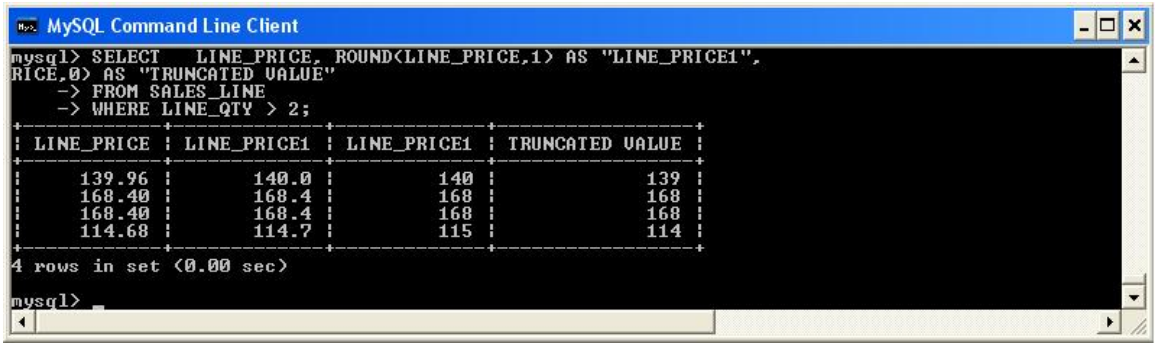
7.2 Numeric Functions

Table: 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;



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

- ✓ SELECT TRANSACTION_NO, LINE_PRICE, MOD(LINE_PRICE, 10)
  FROM SALES_LINE
  WHERE LINE_QTY > 2;

7.3 String Functions

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

Table: 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)

CONCAT

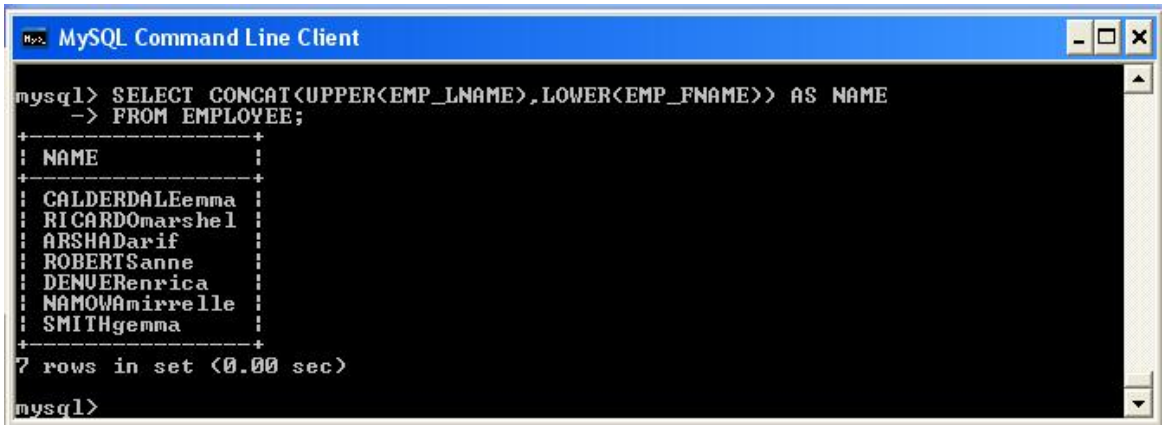
The following query illustrates the CONCAT function. It lists all employee first and last names concatenated together. The output for this query can be seen in Figure

- ✓ SELECT CONCAT(EMP_LNAME ,EMP_FNAME) AS NAME
  FROM EMPLOYEE;

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.

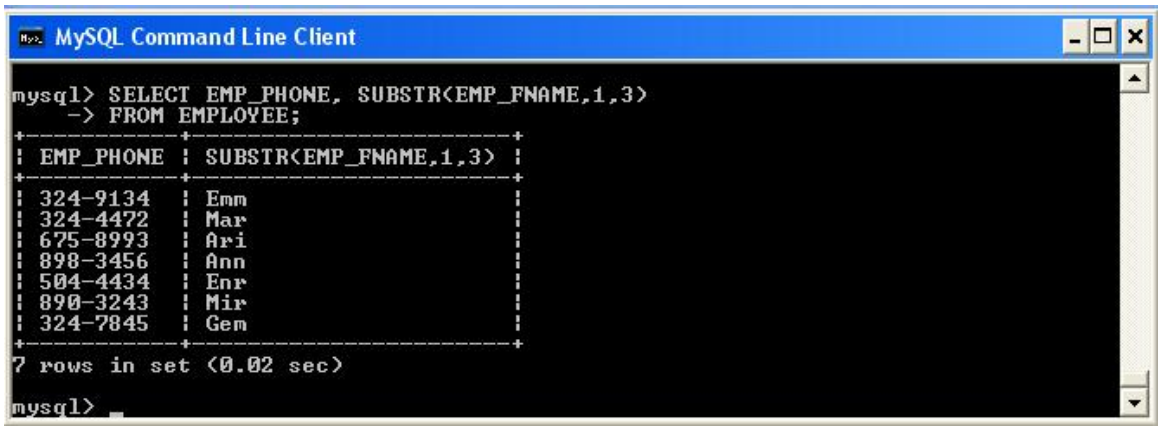
- ✓ SELECT CONCAT(UPPER(EMP_LNAME),LOWER(EMP_FNAME)) AS NAME
  FROM EMPLOYEE;



SUBSTR

The following example lists the first three characters of all the employees' first name. The output of this query is shown in Figure.

- ✓ SELECT EMP_PHONE, SUBSTR(EMP_FNAME,1,3)
  FROM EMPLOYEE;

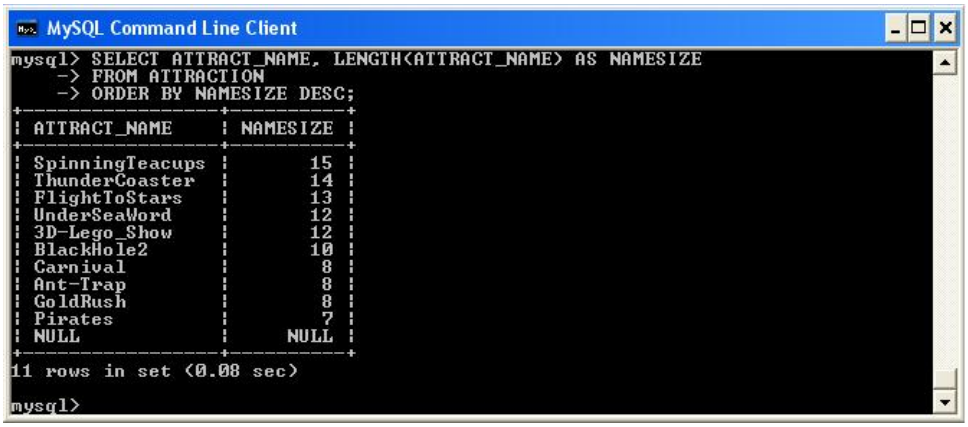


Task 7.10 Write a query which generates a list of employee user IDs, using the first day of the month they were born and the first six characters of last name in UPPER case. Your query should return the results shown in Figure.

LENGTH

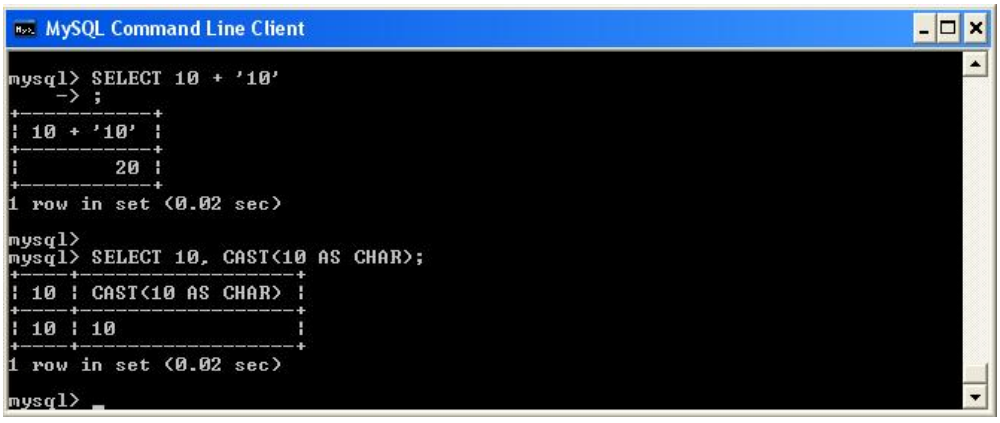
The following example lists all attraction names and the length of their names; ordered descended by attraction name length. The output of this query is shown in Figure 6.

- ✓ SELECT ATTRACT_NAME, LENGTH(ATTRACT_NAME) AS NAMESIZE
  FROM ATTRACTION
  ORDER BY NAMESIZE DESC;



7.4 Conversion Functions

Conversion functions allow you to take a value of a given data type and convert it to the equivalent value in another data type. In MySQL, some conversions occur implicitly. For example, MySQL automatically converts numbers to strings when needed, and vice versa.



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 useful for avoiding errors caused by incorrect calculation when one of the arguments is null.

Task 7.11

1. Load and run the script *sales_copy.sql*

```
CREATE TABLE SALES_COPY (
TRANSACTION_NO NUMERIC(8),
LINE_NO NUMERIC(2),
ITEM_PRICE NUMERIC(2,0),
LINE_QTY NUMERIC(4),
LINE_PRICE NUMERIC(9,2),
CONSTRAINT PK_SALES_COPY_LINE PRIMARY KEY (TRANSACTION_NO,LINE_NO));

INSERT INTO SALES_COPY VALUES (10000001,1,10.99,2,21.98);
INSERT INTO SALES_COPY VALUES (10000001,2,15.00,2,30.00);
INSERT INTO SALES_COPY VALUES (10000002,1,10.99,1,10.99);
INSERT INTO SALES_COPY VALUES (10000002,2,20.99,1,20.99);
INSERT INTO SALES_COPY VALUES (10000003,1,20.99,NULL,NULL);
INSERT INTO SALES_COPY VALUES (10000003,2,15.00,NULL,NULL);
INSERT INTO SALES_COPY VALUES (10000004,1,15.00,2,30.00);
INSERT INTO SALES_COPY VALUES (10000004,2,20.99,2,41.98);
```

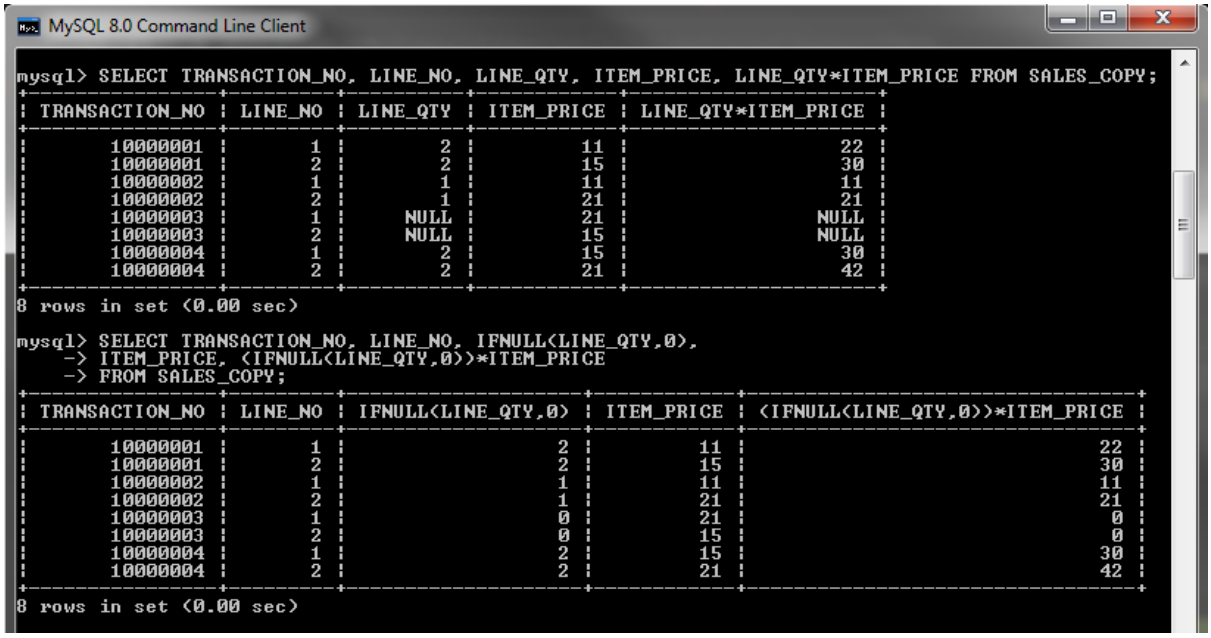
2. Enter the following query which displays to the screen the Total of the LINE_QTY * LINE_PRICE.

   - ✓ SELECT TRANSACTION_NO, LINE_NO, LINE_QTY, ITEM_PRICE, LINE_QTY*ITEM_PRICE AS "TOTAL SALES PER LINE" FROM SALES_COPY;

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

   - ✓ SELECT TRANSACTION_NO, LINE_NO,

IFNULL(LINE_QTY,0),ITEM_PRICE,(IFNULL(LINE_QTY,0))*ITEM_PRICE
AS"TOTAL SALES PER LINE"
FROM SALES_COPY;



CASE

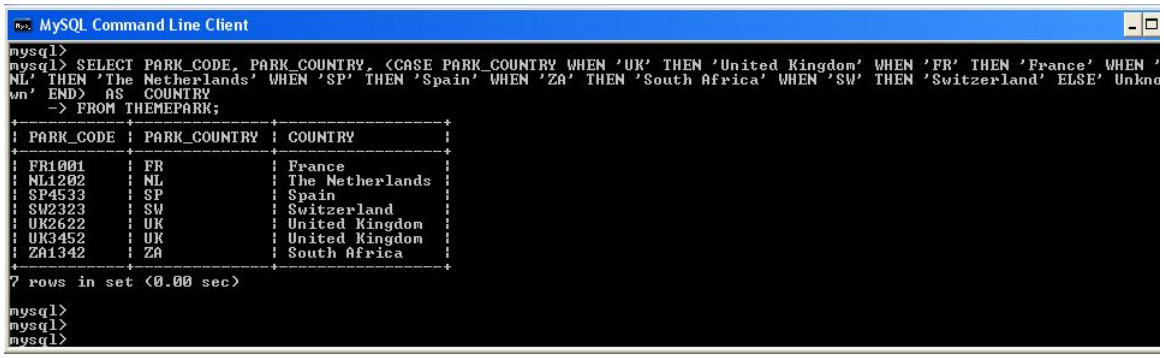
The CASE function compares an attribute or expression with a series of values and returns an associated value or a default value if no match is found. There are two versions of the CASE function. The syntax of each is shown below.

- ✓ 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

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'.

- ✓ 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 Command Line Client
mysql>
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 'Unkno
wn' 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>

Exercises Lab 7

Exercise 7.1 Write a query which lists the names and dates of births of all employees born on the 14th day of the month.

Exercise 7.2 Write a query which lists the approximate age of the employees on the company's tenth anniversary date (11/25/2008).

Exercise 7.3 Write a query which generates a list of employee user passwords, using the first three digits of their phone number, and the first two characters of first name in lower case. Label the column USER_PASSWORD;

Exercise 7.4 Write a query which displays the last date a ticket was purchased in all Theme Parks. You should also display the Theme Park name. Print the date in the format 12th January 2007.

Lab 8: Subqueries

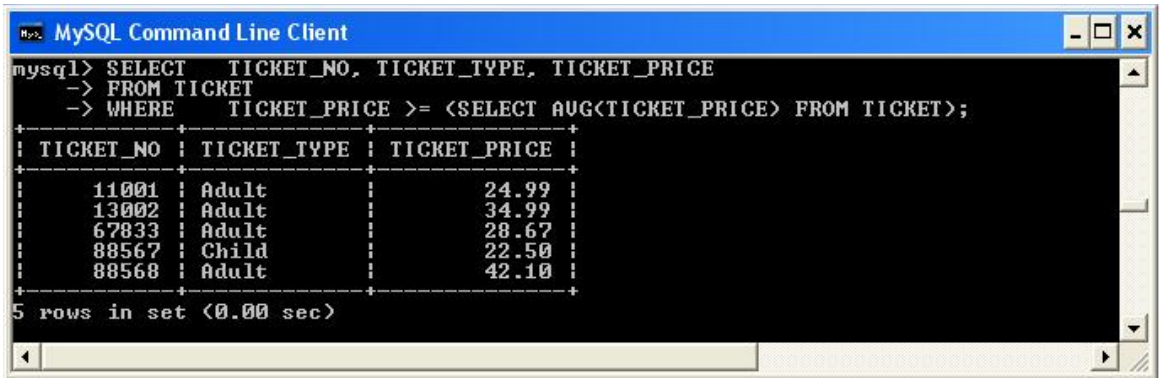
The learning objectives of this lab are to

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

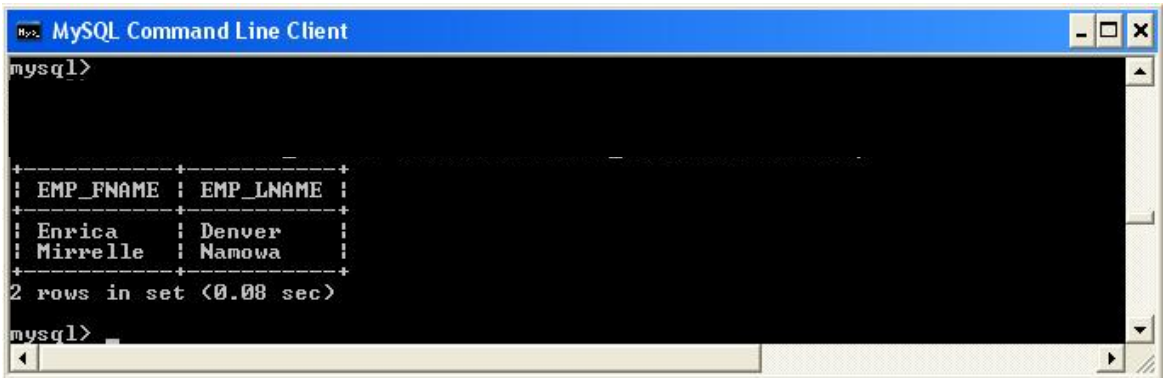
8.1 SELECT Subqueries

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);
```



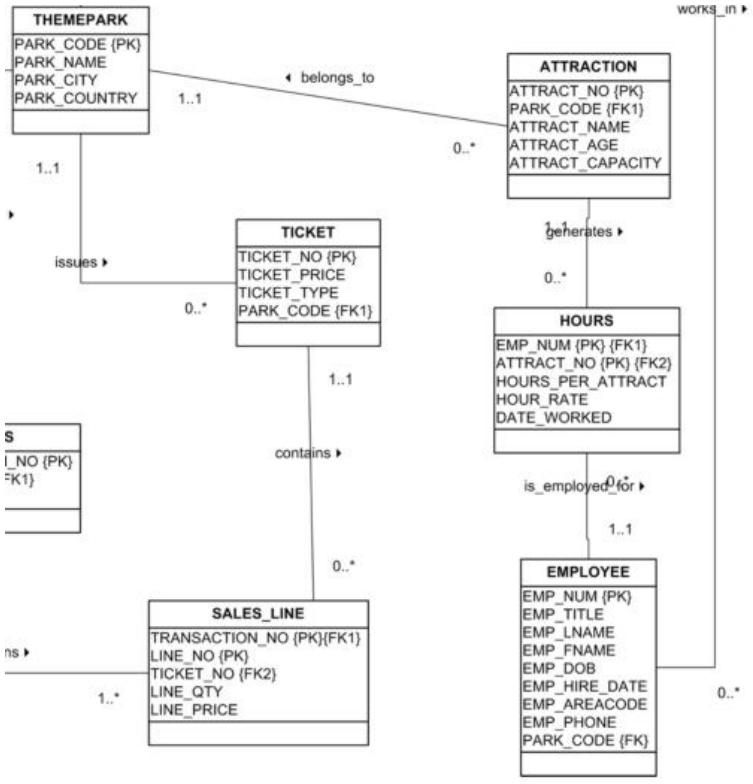
Task 8.1 Write a query that displays the first name, last name of all employees who earn more than the average hourly rate. Do not display duplicate rows. Your output should match that shown in Figure.



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.

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

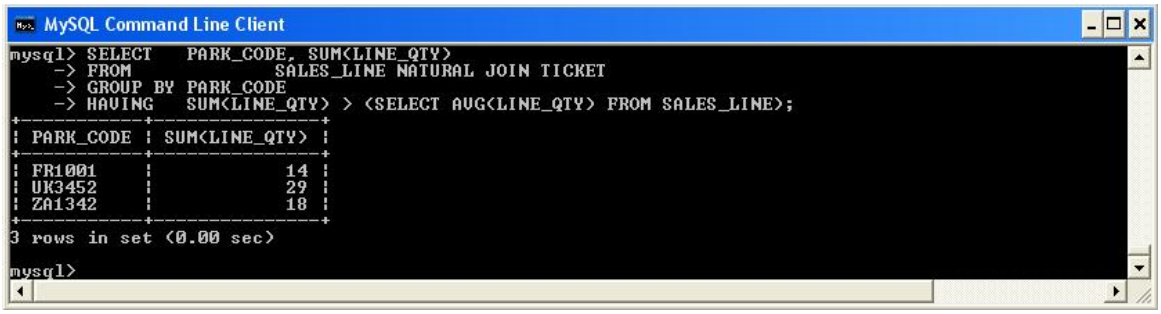
- ✓ SELECT DISTINCT EMP_NUM, EMP_LNAME, EMP_FNAME, PARK_NAME
  FROM EMPLOYEE NATURAL JOIN HOURS NATURAL JOIN
  ATTRACTION NATURAL JOIN THEMEPARK
  WHERE PARK_CODE IN (SELECT THEMEPARK.PARK_CODE FROM
  THEMEPARK WHERE PARK_NAME LIKE '%Fairy%');



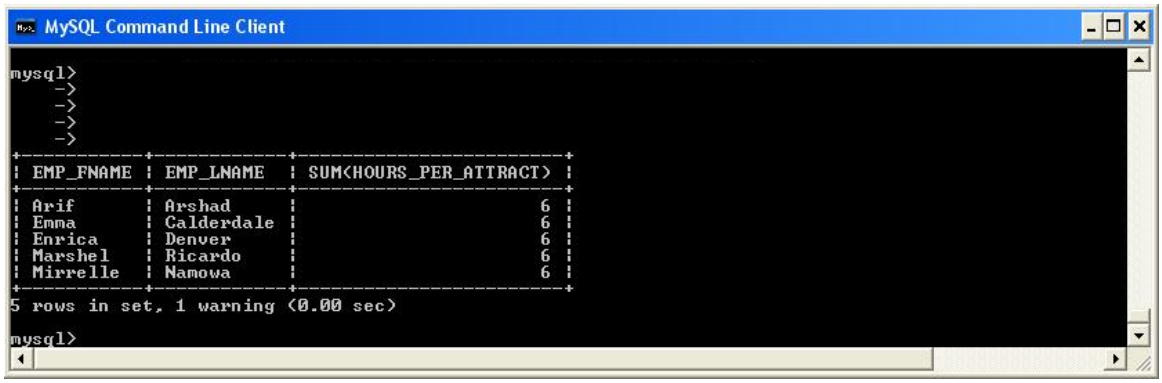
8.3 HAVING Subqueries

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);



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 greater than the average number of hours in total during May 2007. Your output should match that shown in Figure.



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?

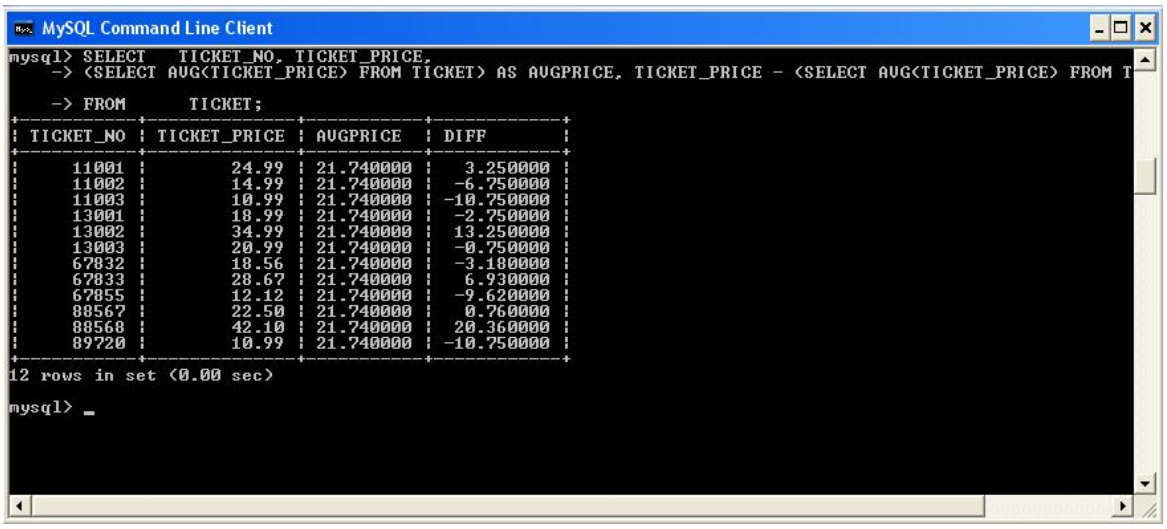
For example, to find the ticket_numbers and corresponding park_codes of the tickets that are priced higher than the highest-priced 'Child' ticket you could write the following query.

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

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;



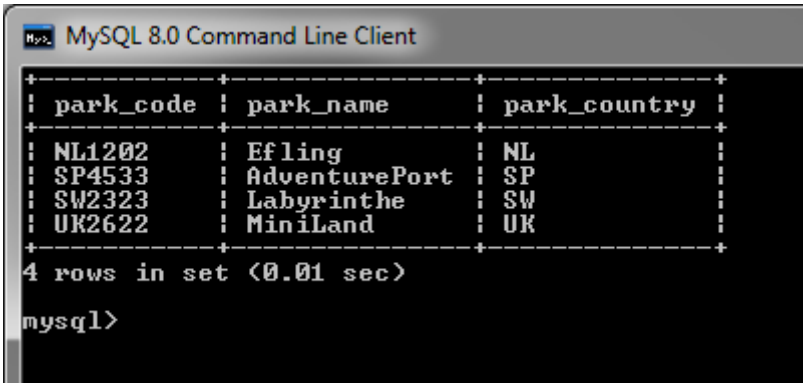
8.6 Correlated Subqueries

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

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);

Task 8.4 Modify the above query to display all the theme parks where there have been no recorded tickets sales recently.



Lab 9: Views

The learning objectives of this lab are to

- Create a simple view
- Manage database constraints in views using the WITH CHECK OPTION

9.1 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:

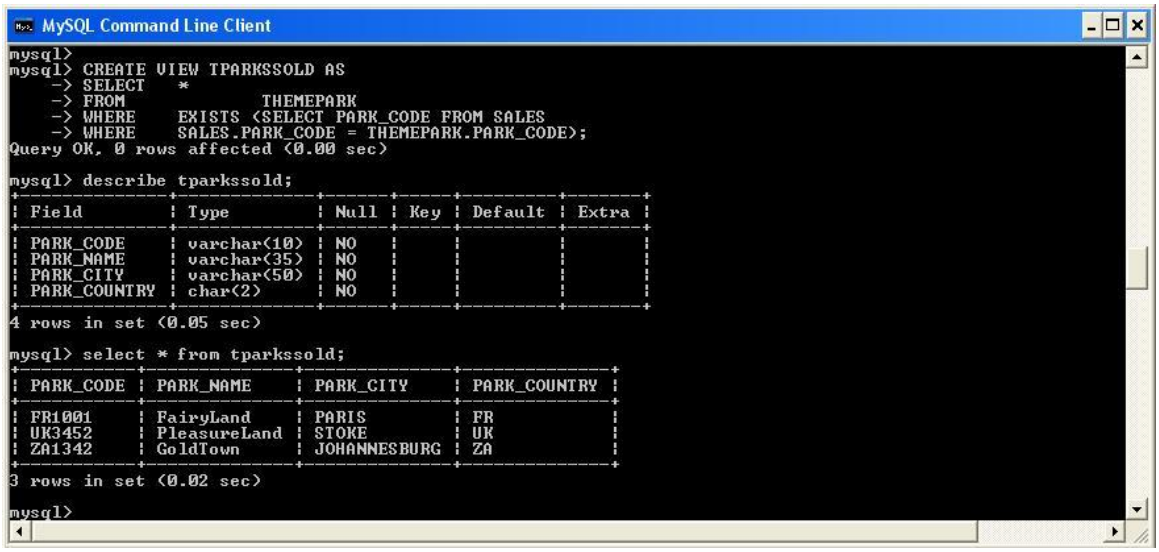
- ✓ CREATE VIEW *viewname* AS SELECT *query*

For example, to create a view of only those Theme Parks were tickets have been sold you would do so as follows:

- ✓ CREATE VIEW TPARKSSOLD AS
  SELECT * FROM THEMEPARK
  WHERE EXISTS (SELECT PARK_CODE FROM SALES
  WHERE SALES.PARK_CODE = THEMEPARK.PARK_CODE);

To display the contents of this view you would type

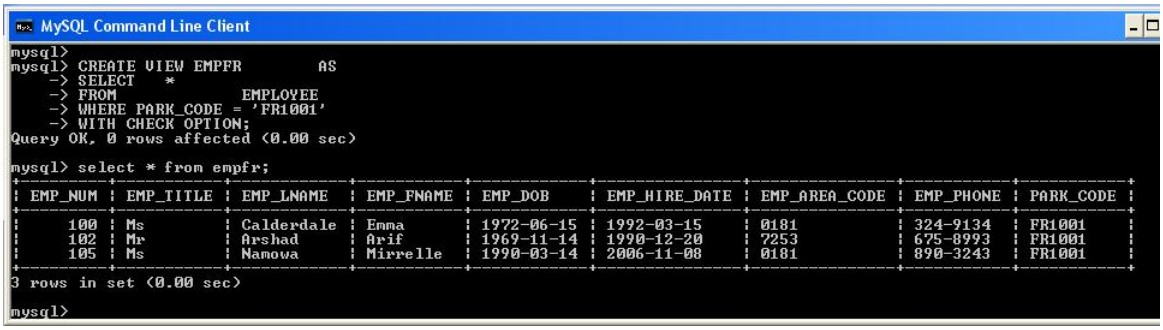
- ✓ SELECT * FROM TPARKSSOLD;



9.2 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.

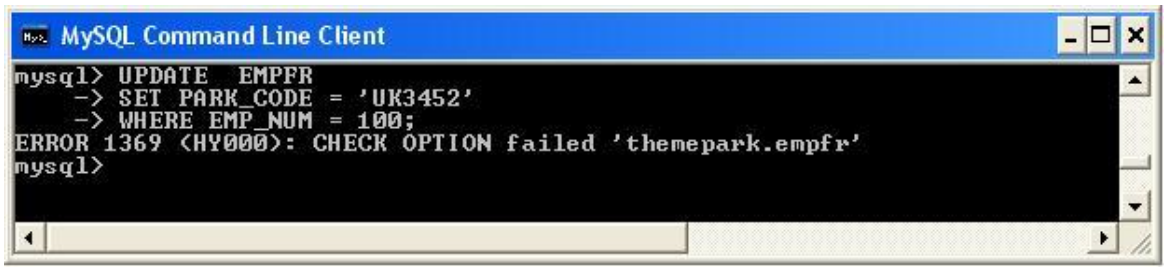
- ✓ CREATE VIEW EMPFR AS
  SELECT *
  FROM EMPLOYEE
  WHERE PARK_CODE = 'FR1001'
  WITH CHECK OPTION;



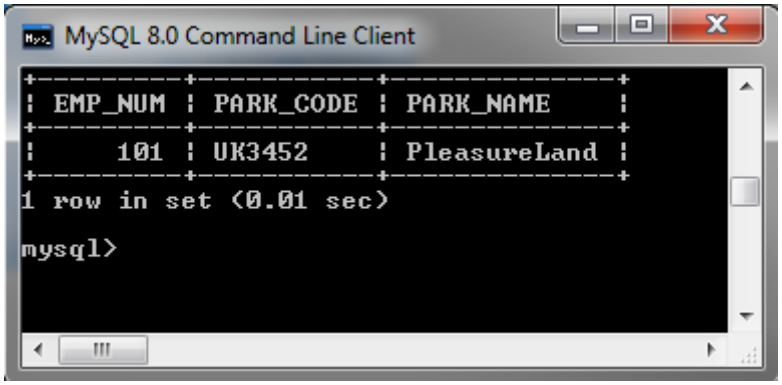
So for example if employee 'Emma Caulderdale' was to leave the park and move to park 'UK3452', we would want to update her information with the following query:

- ✓ UPDATE EMPFR
  SET PARK_CODE = 'UK3452'
  WHERE EMP_NUM = 100;

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



Task 9.1 Create the view EMPFR and tray and update the Theme Park that employee number 101 works in.



Task 9.2 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.

Task 9.3 Remove the EMPFR view.

Exercises Lab 9

Exercise 9.1 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.*

Exercise 9.2 Check that the view works, by displaying its contents.

Exercise 9.3 Using your view EMP_DETAILS, write a query that displays all employee first and last names and the park names.

Exercise 9.4 Remove the view EMPDETAILS.