Database Systems Lab

Lab # 02

Building a database Table by Table
And Data Manipulation Commands

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

2.1 Introduction

In this section you will learn how to create a small database called Theme Park from the ERD shown in Figure 11. This will involve you creating the table structures in MySQL using the CREATE TABLE command. In order to do this, appropriate data types will need to be selected from the data dictionary for each table structure along with any constraints that have been imposed (e.g. primary and foreign key). Converting any ER model to a set of tables in a database requires following specific rules that govern the conversion. The application of those rules requires an understanding of the effects of updates and deletions on the tables in the database. You can read more about these rules in Chapter 8, Introduction to Structured Query Language, and Appendix D, Converting an ER Model into a Database Structure.

2.2 The Theme Park Database

Figure 11 shows the ERD for the Theme Park database which will be used throughout this lab guide.

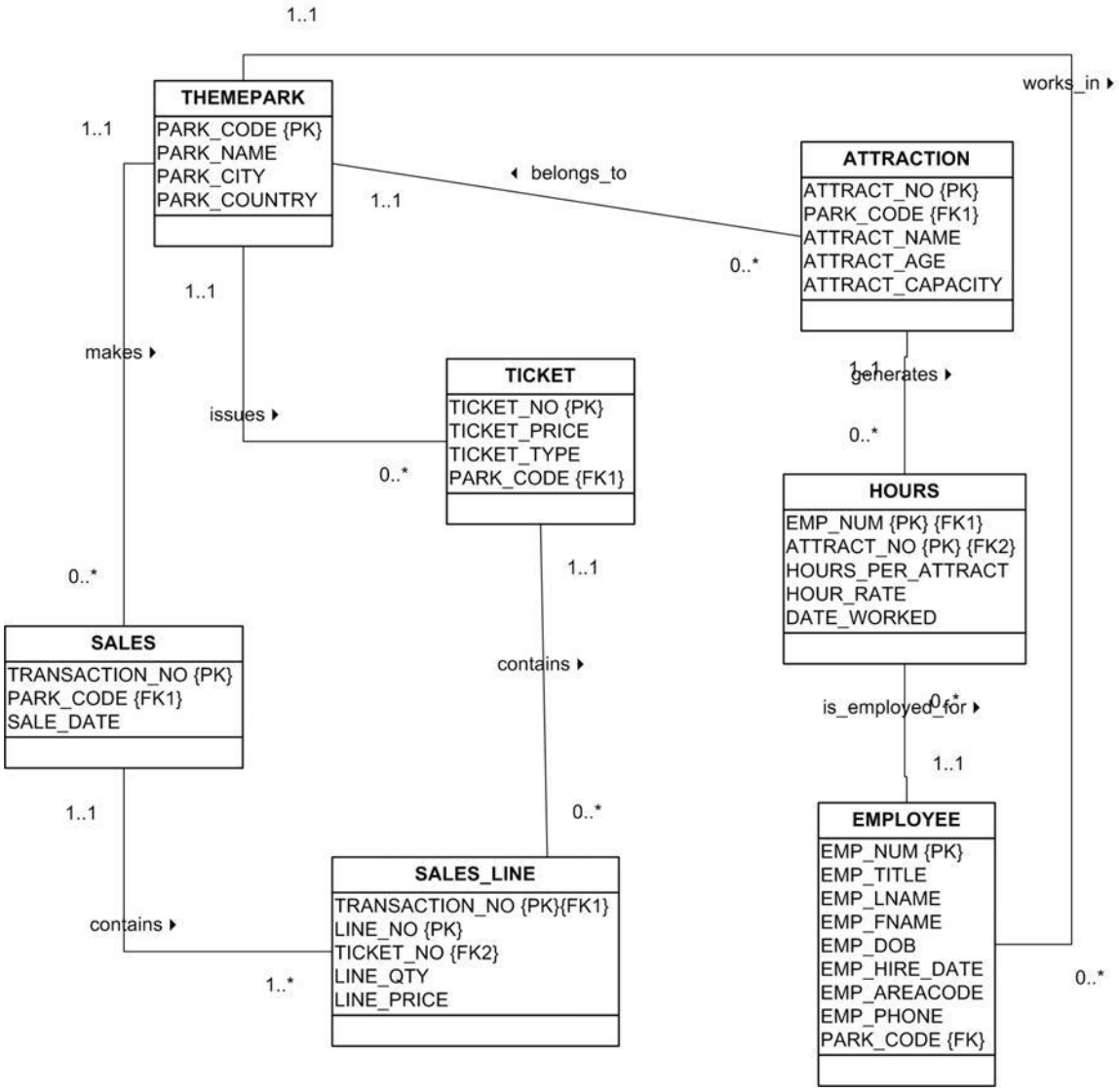

Figure 11 The Theme park Database ERD

Table 2.1 Shows the Data Dictionary for the Theme Park database which will be used to create each table structure.

Table 2.1 Data Dictionary for the Theme Park Database

Table Name	Attribute Name	Contents	Data Type	Format	Range	Required	PK or FK	FK Referenced Table
THEMEPARK	PARK_CODE	Park code	VARCHAR(10)	XXXXXXXX	NA	Y	PK	
	PARK_NAME	Park Name	VARCHAR(35)	XXXXXXXX	NA	Y		
	PARK_CITY	City	VARCHAR(50)		NA	Y		
	PARK_COUNTRY	Country	CHAR(2)	XX	NA	Y		
EMPLOYEE	EMP_NUM	Employee number	NUMERIC(4)	##	0000 – 9999	Y	PK	
	EMP_TITLE	Employee title	VARCHAR(4)	XXXX	NA	N		
	EMP_LNAME	Last name	VARCHAR(15)	XXXXXXXX	NA	Y		
	EMP_FNAME	First Name	VARCHAR(15)	XXXXXXXX	NA	Y		
	EMP_DOB	Date of Birth	DATE	DD-MON-YY	NA	Y		
	EMP_HIRE_DATE	Hire date	DATE	DD-MON-YY	NA	Y		
	EMP_AREACODE	Area code	VARCHAR(4)	XXXX	NA	Y		
	EMP_PHONE	Phone	VARCHAR (12)	XXXXXXXX	NA	Y		
	PARK_CODE	Park code	VARCHAR(10)	XXXXXXXX	NA	Y	FK	THEMEPARK

TICKET	TICKET_NO	Ticket number	NUMERIC(10)	##########	NA	Y		
	TICKET_PRICE	Price	NUMERIC(4,2)	####.##	0.00 – 0000.00			
	TICKET_TYPE	Type of ticket	VARCHAR(10)	XXXXXXXXXX	Adult, Child,Senior,Other			
	PARK_CODE	Park code	VARCHAR(10)	XXXXXXXX	NA	Y	FK	THEMEPARK
ATTRACTION	ATTRACT_NO	Attraction number	NUMERIC(10)	##########	N/A	Y	PK	
	PARK_CODE	Park code	VARCHAR(10)	XXXXXXXX	NA	Y	FK	THEMEPARK
	ATTRACT_NAME	Name	VARCHAR(35)	XXXXXXX	N/A	N		
	ATTRACT_AGE	Age	NUMERIC(3)	###	Default 0	Y		
	ATTRACT_CAPACITY	Capacity	NUMERIC(3)	###	N/A	Y		
HOURS	EMP_NUM	Employee number	NUMERIC(4)	##	0000 – 9999	Y	PK / FK	EMPLOYEE
	ATTRACT_NO	Attraction number	NUMERIC(10)	##########	N/A	Y	PK / FK	ATTRACTION
	HOURS_PER_ATTRACT	Number of hours	NUMERIC(2)	##	N/A	Y		
	HOUR_RATE	Hourly Rate	NUMERIC(4,2)	####.##	N/A	Y		
	DATE_WORKED	Date worked	DATE	DD-MON-YY	N/A	Y		

SALES	TRANSACTION_NO	Transaction No	NUMERIC	##########	N/A	Y	PK	
	PARK_CODE	Park code	VARCHAR(10)	XXXXXXXX	NA	Y	FK	THEMEPARK
	SALE_DATE	Date of Sale	DATE	DD-MON-YY	SYSDATE	Y		
SALESLINE	TRANSACTION_NO	Transaction No	NUMERIC	##########	N/A	Y	PK / FK	SALES
	LINE_NO	Line number	NUMERIC(2)	##	N/A	Y		
	TICKET_NO	Ticket number	NUMERIC(10)	#########	NA	Y	FK	TICKET
	LINE_QTY	Quantity	NUMERIC(4)	####	N/A	Y		
	LINE_PRICE	Price of line	NUMERIC(9,2)	#########.##	N/A	Y		

2.3 Data Types in MySQL

In order to build tables in MySQL you will need to specify the data type for each column. Table 2.2 shows some of the most common data types. If you have previously used an ORACLE DBMS, you will notice that the syntax is different.

Table 2.2 Common MySQL data types[1]

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)	No trailing spaces are appended to the end of this datatype.

[1] This table was adapted from the web site http://www.developerfusion.co.uk/. A comprehensive and complete list of types can be taken from the MySQL Reference Manual.

		MySQL keeps track of a delimiter to keep track of the end of the field.
TINYTEXT	fieldName TINYTEXT	Stores up to 255 characters. Equivalent to VARCHAR(255).
TEXT	fieldName TEXT	Stores up to 65,535 characters. An Index can be created on the first 255 characters of a field with this data type.
MEDIUMTEXT	fieldName MEDIUMTEXT	Stores up to 16,777,215 characters. An Index can be created on the first 255 characters of a field with this data type.
LONGTEXT	fieldName LONGTEXT	Stores up to 4,294,967,295 characters. An Index can be created on the first 255 characters of a field with this data type. Note: The maximum size of a string in MySQL is currently 16 million bytes, so this data types is not useful at the moment.
ENUM	fieldName ENUM('Yes', 'No')	Stores up to 65,535 enumerated types. The DEFAULT modifier may be used to specify the default value for this field.
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. fieldName INT UNSIGNED The ZEROFILL attribute may be used to left-pad any of the integer with zero's. fieldName INT ZEROFILL The AUTO_INCREMENT attribute may be used with any of the Integer data types. The following example could be used to create a primary key using the AUTO_INCREMEMNT attribute. fieldName INT UNSIGNED AUTO_INCREMENT PRIMARY KEY
TINYINT	fieldName TINYINT	Stores a signed or unsigned byte. Unsigned bytes have a range of 0 to 255, and signed bytes have a range of -128 to 127. By default, the TINYINT data type is signed.
MEDIUMINT	fieldName MEDIUMINT	Stores a signed or unsigned medium sized integer. Unsigned fields of this type have a range of 0 to 1,677,215, and signed fields of this type have a range of -8,388,608 to 8,388,607. By default, the MEDIUMINT data type is signed.
BIGINT	fieldName BIGINT	Stores a signed or unsigned big integer. Unsigned fields of this type have a range of 0 to 18,446,744,073,709,551,615, and signed fields of this type have a range of -9,223,372,036,854,775,808 to 9,223,327,036,854,775,807. By default, the BIGINT data type is signed.

FLOAT	fieldName FLOAT	Used for single precision floating point numbers.
DOUBLE	fieldName DOUBLE	Used for double precision floating point numbers.
DATE	fieldName DATE	Stores dates in the format YYYY-MM-DD.
TIMESTAMP(size)	fieldName DATETIME	Stores dates and times in the format YYYY-MM-DD HH:MM:SS. Automatically keeps track of the time the record was last ammended. The following table shows the formats depending on the size of TIMESTAMP
DATETIME	fieldName TIMESTAMP(14)	
TIME	fieldName TIME	Stores times in the format HH:MM:SS.
YEAR(size)	fieldName YEAR(4)	Stores the year as either a 2 digit number, or a 4 digit number, depending on the size provided.

Size	Format
2	YY
4	YYMM
6	YYMMDD
8	YYYYMMDD
10	YYYYMMDDHH
12	YYYYMMDDHHMM
14	YYYYMMDDHHMMSS

2.4 Creating the Table Structures

Use the following SQL commands to create the table structures for the Theme Park database. Enter each one separately to ensure that you have no errors. Successful table creation will prompt MySQL to say “Query OK”. It is useful to store each correct table structure in a script file, in case the entire database needs to be recreated again later. You can use a simple text editor such as notepad in order to do this. Save the file as themepark.sql. Note that the table-creating SQL commands used in this example are based on the data dictionary shown in Table 2.1 and the MySQL data types in Table 2.2.

As you examine each of the SQL table-creating command sequences in the following tasks, note the following features:

- The NOT NULL specifications for the attributes ensure that a data entry will be made. When it is crucial to have the data available, the NOT NULL specification will not allow the end user to leave the attribute empty (with no data entry at all).
- The UNIQUE specification creates a unique index in the respective attribute. Use it to avoid duplicated values in a column.
- The primary key attributes contain both a NOT NULL and a UNIQUE specification. Those specifications enforce the **entity integrity (***entity integrity ensures that there are no duplicate records within the table and that the field that identifies each record within the table is unique and never null. The existence of the Primary Key is the core of the entity integrity. If you define a primary key for each entity, they follow the entity integrity rule.***)** requirements. If the NOT NULL and UNIQUE specifications are not supported, use PRIMARY KEY without the specifications.
- The entire table definition is enclosed in parentheses. A comma is used to separate each table element (attributes, primary key, and foreign key) definition.
- The DEFAULT constraint is used to assign a value to an attribute when a new row is added to a table. The end user may, of course, enter a value other than the default value. In MYSQL the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as the system date like you can do in an ORACLE DBMS.

- The FOREIGN KEY CONSTRAINT is used to enforce referential integrity (*referential integrity constraint is specified between two tables*). In order to set up a foreign key relationship between two MySQL tables, three conditions must be met:

    1. Both tables must be of the InnoDB table type.
    2. The fields used in the foreign key relationship must be indexed.
    3. The fields used in the foreign key relationship must be similar in data type.

2.4.1 Creating the THEMEPARK Database.

Task 2.1 At the MySQL prompt; create a database called Theme Park as shown in Lab 1. Then select the database for use as shown in Figure 12.

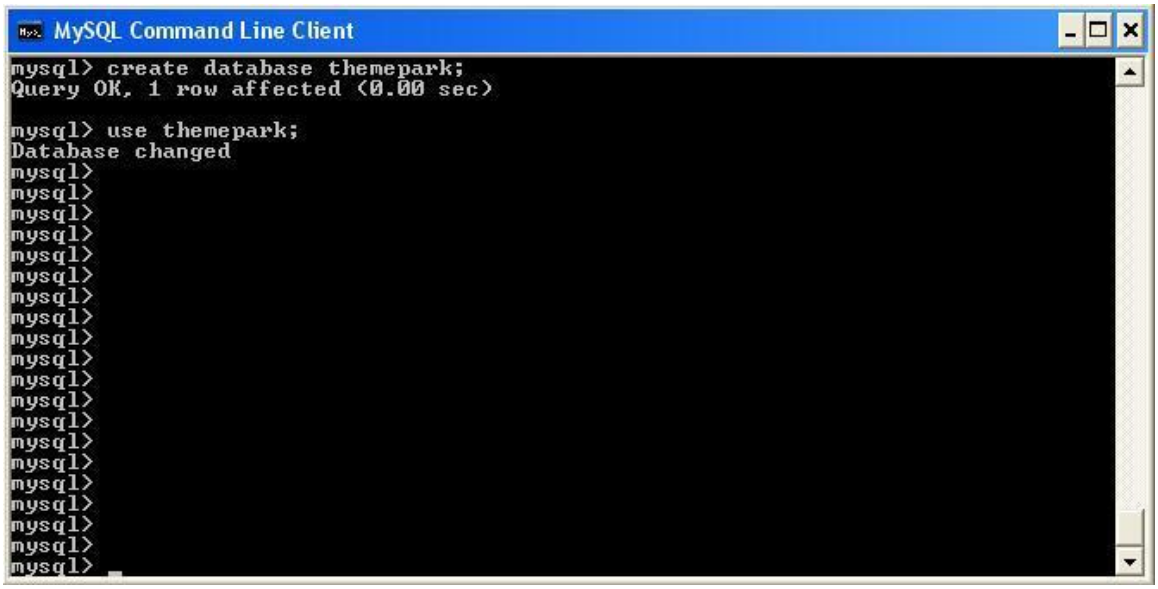

Figure 12 Creating and using the Theme Park Database.

2.4.2 Creating the THEMEPARK TABLE

Task 2.2 Enter the following SQL command to create 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);
```

Notice that when you create the THEMEPARK table structure you set the stage for the enforcement of entity integrity rules by using:

```
PARK_CODE        VARCHAR(10) PRIMARY KEY,
```

As you create this structure, also notice that the NOT NULL constraint is used to ensure that the columns PARK_NAME, PARK_CITY and PARK_COUNTRY does not accept nulls.

Remember to store this CREATE TABLE structure in your themepark.sql script.

2.4.3 Creating the EMPLOYEE TABLE

Task 2.3 Enter the following SQL command to create 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),
INDEX           (PARK_CODE),
CONSTRAINT      FK_EMP_PARK FOREIGN KEY(PARK_CODE) REFERENCES

THEMEPARK(PARK_CODE));
```

As you look at the CREATE TABLE sequence, note that referential integrity has been enforced by specifying a constraint called FKP_EMP_PARK. In order to use foreign key constraints in MySQL, notice that the PARK_CODE column is first indexed. This foreign key constraint definition ensures that you cannot delete a Theme Park from the THEMEPARK table if at least one employee row references that Theme Park and that you cannot have an invalid entry in the foreign key column.

Remember to store this CREATE TABLE structure in your themepark.sql script.

2.4.4 Creating the TICKET TABLE

Task 2.4 Enter the following SQL command to create 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),
INDEX             (PARK_CODE),
CONSTRAINT        FK_TICKET_PARK FOREIGN KEY(PARK_CODE)
REFERENCES THEMEPARK(PARK_CODE));
```

As you create the TICKET table, notice that both PRIMARY and FOREIGN KEY constraints have been applied. Remember to store this CREATE TABLE structure in your themepark.sql script.

2.4.5 Creating the ATTRACTION TABLE

Task 2.5 Enter the following SQL command to create 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),
INDEX                   (PARK_CODE),
```

CONSTRAINT FK_ATTRACT_PARK FOREIGN KEY(PARK_CODE)

REFERENCES THEMEPARK(PARK_CODE));

Remember to store this CREATE TABLE structure in your themepark.sql script.

2.4.6 Creating the HOURS TABLE

Task 2.6 Enter the following SQL command to create 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,
INDEX                   (EMP_NUM),
INDEX                   (ATTRACT_NO),
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));
```

As you create the HOURS table, notice that the HOURS table contains FOREIGN KEYS to both the ATTRACTION and the EMPLOYEE table.

Remember to store this CREATE TABLE structure in your themepark.sql script.

2.4.7 Creating the SALES TABLE

Task 2.7 Enter the following SQL command to create the SALES table.

```
CREATE TABLE SALES (
TRANSACTION_NO      NUMERIC PRIMARY KEY,
PARK_CODE           VARCHAR(10),
SALE_DATE           DATE NOT NULL,
INDEX               (PARK_CODE),
CONSTRAINT      FK_SALES_PARK FOREIGN KEY(PARK_CODE)
```

REFERENCES THEMEPARK(PARK_CODE));

Remember to store this CREATE TABLE structure in your themepark.sq script.

2.4.8 Creating the SALESLINE TABLE

Task 2.8 Enter the following SQL command to create 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,
INDEX                   (TRANSACTION_NO),
INDEX                   (TICKET_NO),
CONSTRAINT      PK_SALES_LINE   PRIMARY KEY
(TRANSACTION_NO,LINE_NO),
CONSTRAINT      FK_SALES_LINE_SALES FOREIGN KEY
(TRANSACTION_NO) REFERENCES SALES(TRANSACTION_NO),
CONSTRAINT      FK_SALES_LINE_TICKET FOREIGN KEY (TICKET_NO)
REFERENCES TICKET(TICKET_NO));
```

Remember to store this CREATE TABLE structure in your themepark.sq script.

The **DROP TABLE** command permanently deletes a table (and thus its data) from the database schema. When you write a script file to create a database schema, it is useful to add DROP TABLE commands at the start of the file. If you need to amend the table structures in any way, just one script can then be run to re-create all the database structures. Primary and foreign key constraints control the order in which you drop the tables – generally you drop in the reverse order of creation. The DROP commands for the Theme Park database are:

```
DROP TABLE SALES_LINE;

DROP TABLE SALES;
DROP TABLE HOURS;
DROP TABLE ATTRACTION;
DROP TABLE TICKET;
DROP TABLE EMPLOYEE;
DROP TABLE THEMEPARK;
```

Task 2.9. Add the DROP commands to the start of your script file and then run the themepark.sql script.

2.5 Display a table's structure

The command **DESCRIBE** is used to display the structure of an individual table. To see the structure of the EMPLOYEE table you would enter the command: DESCRIBE EMPLOYEE as shown in Figure 13.

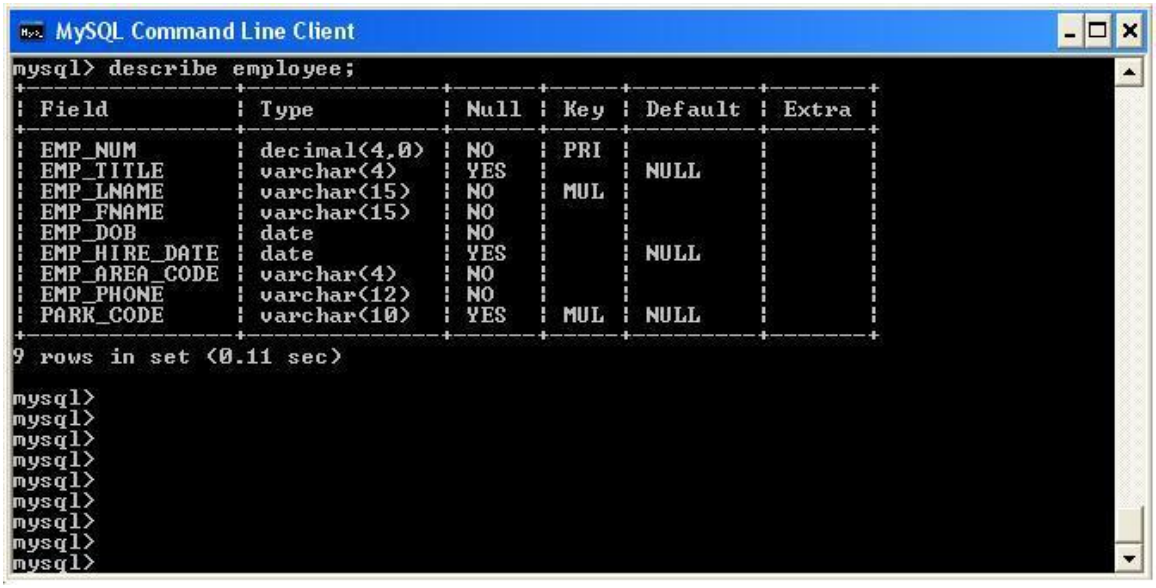

Figure 13 Describing the structure of the THEMEPARK Table

Task 2.10 Use the DESCRIBE command to view the structure of the other database tables that you have created in this lab.

2.6 Listing all tables

Task 2.11 Use the SHOW TABLES command as shown in Figure 14, to list all tables that have been created within the THEMEPARK database.

Figure 14 Displaying all tables

2.7 Altering the table structure

All changes in the table structure are made by using the **ALTER TABLE** command, followed by a keyword that produces the specific change you want to make. Three options are available: ADD, MODIFY, and DROP. ADD enables you to add a column, and MODIFY enables you to change column characteristics. Most RDBMSs do not allow you to delete a column (unless the column does not contain any values) because such an action may delete crucial data that are used by other tables.

Supposing you wanted to modify the column ATTRACT_CAPACITY in the ATTRACTION table by changing the date characteristics from NUMERIC(3) to NUMERIC(4). You would execute the following command:

ALTER TABLE ATTRACTION

MODIFY ATTRACT_CAPACITY NUMERIC(4);

The tables that you have created will be used in the rest of this lab guide to explore the use of SQL in MySQL in more detail.

3.1 Adding Table Rows

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*).

The order in which you insert data is important. For example, because the TICKET uses its PARK_CODE to reference the THEMEPARK table's PARK_CODE, an integrity violation will occur if those THEMEPARK table PARK_CODE values don't yet exist.

Therefore, you need to enter the THEMEPARK rows before the TICKET rows. Complete the following tasks to insert data into the THEMEPARK and TICKET tables:

Task 2.12 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 2.13 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');

Task 2.14 Run the script file **themeparkdata.sql** to insert the rest of the data into the Theme Park database.

3.2 Retrieving data from a table using the SELECT Statement

Select is used to retrieve rows selected from one or more tables.

The SELECT command has many optional clauses but in its simplest can be written as

SELECT *columnlist*

FROM *tablelist*

[WHERE *conditionlist*];

Notice that the command must finish with a semi-colon, and will be executed when the Enter key is pressed at the end of the command.

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 15.

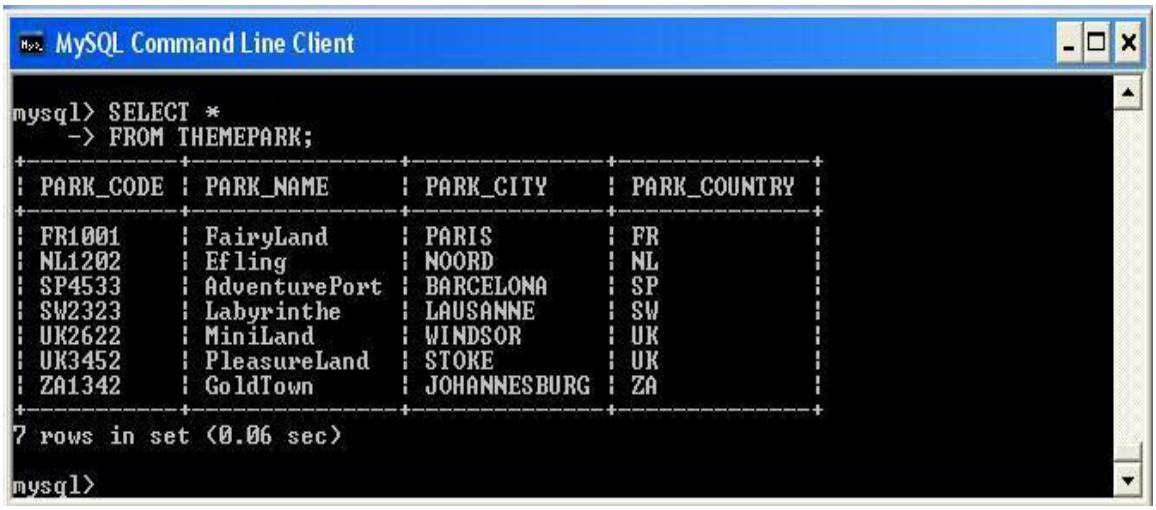

Figure 15: Displaying all columns from the THEMEPARK Table

The SELECT command and the FROM clause are necessary for any SQL query, and must always be included so that the DBMS knows which columns we want to display and which table they come from.

Task 2.15. Type in the following examples of the SELECT statement and check your results with those provided in Figures 16 and 17. In these two examples you are selecting specific columns from a single table.

Example 1

SELECT ATTRACT_NO, ATTRACT_NAME, ATTRACT_CAPACITY

FROM ATTRACTION;

Example 2

SELECT EMP_NUM, EMP_LNAME, EMP_FNAME,

EMP_HIRE_DATE FROM EMPLOYEE;

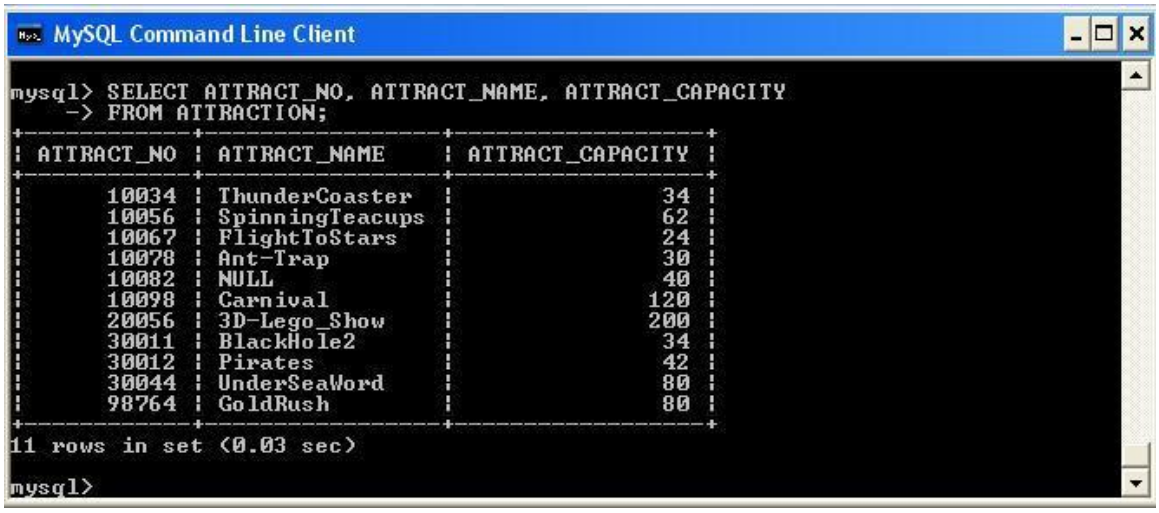

Figure 16: Output for Example 1

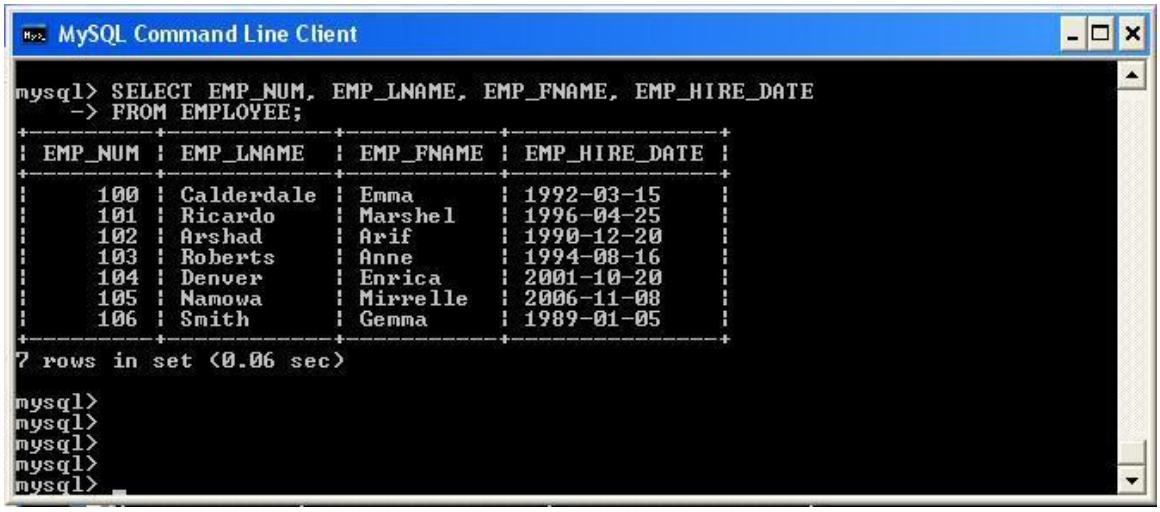

Figure 17: Output for Example 2

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 18.

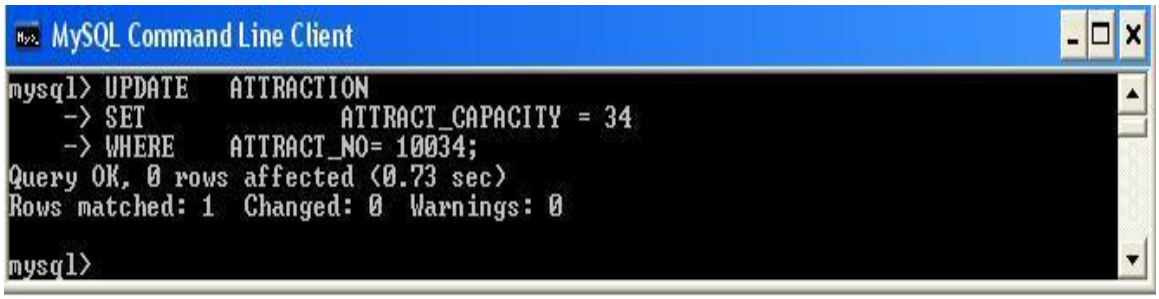

Figure 18: Updating the attraction capacity

> ***Note***
>
> If more than one attribute is to be updated in the row, separate each attribute with commas.

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 2.16 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;
```

Notice that all the values of ATTRACT_AGE have the same value.

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!

> ***Note***
>
> If you make a mistake while working through this lab, use the themepark.sql script to re-create the database schema and to insert the sample data.

Task 2.17

Add the following new themeparks to the themeparks Table

PARK_ CODE	PARK_NAME	PARK_CITY	PARK_COUNTRY
AU1001	**SkiWorld**	**AU**	**UK**
GR5001	**RoboLand**	**GR**	**SP**

Task 2.18

Add the new Employee with the following details in the employee Table

Emp Num	Emp Title	Emp Lname	Emp Fname	Emp DOB	Emp hire date	Emp Area Code	Emp Phone	Park Code
2049	Mr	Rahat	Noman	1990-12-20	2015-5-5	7253	502-4934	**AU1001**

Task 2.19

Update the PARK_NAME from **SkiWorld** in the themepark table to **MiniLand**.

Task 2.20

Delete the Theme Park called **RoboLand**.