1 SQL

In this task, you will create a small database called AMUSEMENT PARK from the ER model shown in Figure 1 and later query it to retrieve useful information. This will involve you creating the table structures in SQL Server using the CREATE TABLE command. In order to do this, you are provided with a data directory (see Table 1) that lists appropriate data types for each of the table structure along with any constraints that have been imposed (e.g. primary and foreign key).

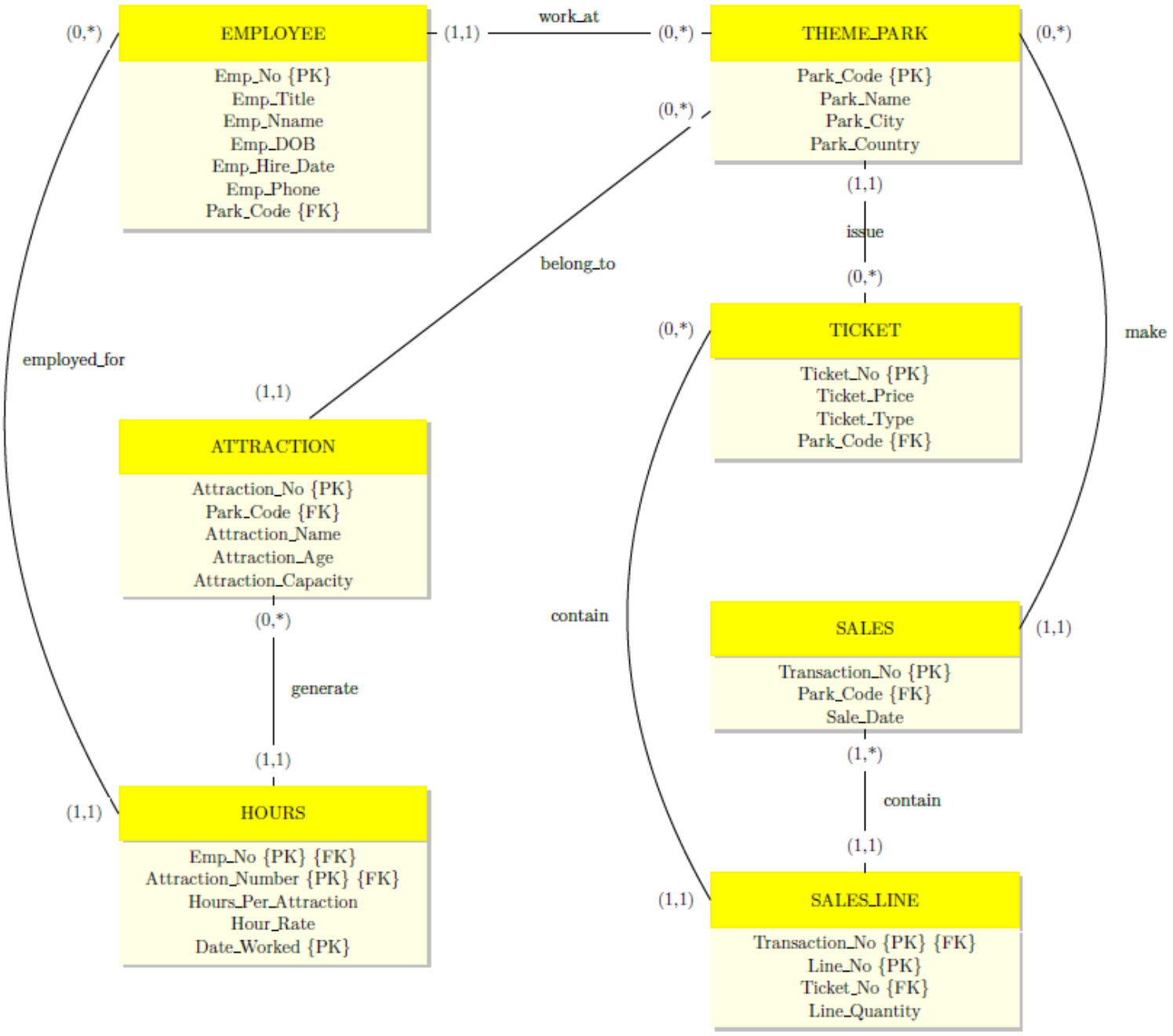


Figure 1: ER Model for AMUSEMENT_PARK Database

Table	Attribute Name	Data Type	Range	Required	PK or FK	PK Referenced Table
THEME_PARK	Park_Code	varchar(10)	N/A	Y	PK	
	Park_Name	varchar(35)	N/A	Y		
	Park_City	varchar(50)	N/A	Y		
	Park_Country	varchar(5)	N/A	Y		
EMPLOYEE	Emp_No	numeric(4)	0000 - 9999	Y	PK	
	Emp_Name	varchar(30)	N/A	Y		
	Emp_Title	varchar(4)	N/A	N		
	Emp_Hire_Date	date	N/A	Y		
	Emp_DOB	date	N/A	Y		
	Emp_Phone	varchar(15)	N/A	Y		
	Park_Code	varchar(10)	N/A	Y	FK	THEME_PARK
TICKET	Ticket_No	numeric(10)	N/A	Y	PK	
	Ticket_Price	float	10.00 - 90.00	Y		
	Ticket_Type	varchar(10)	Child, Adult, Senior,Other	N		
	Park_Code	varchar(10)	N/A	Y	FK	THEME_PARK
ATTRACTION	Attraction_No	numeric(10)	N/A	Y	PK	
	Attraction_Name	varchar(30)	N/A	N		
	Attraction_Age	numeric(3)	N/A	Y		
	Attraction_Capacity	numeric(3)	N/A	Y		
	Park_Code	varchar(10)	N/A	Y	FK	THEME_PARK
HOURS	Emp_No	numeric(4)	0000 - 9999	Y	PK/FK	EMPLOYEE
	Attraction_No	numeric(10)	N/A	Y	PK/FK	ATTRACTION
	Date_Worked	date	N/A	Y	PK	
	Hours_Per_Attraction	numeric(1)	N/A	Y		
	Hour_Rate	float	05.00 - 25.00	Y		
SALES	Transaction_No	numeric(10)	N/A	Y	PK	
	Park_Code	varchar(10)	N/A	Y	FK	THEME_PARK
	Sale_Date	date	N/A	Y		
SALES_LINE	Transaction_No	numeric(10)	N/A	Y	PK/FK	SALES
	Line_No	numeric(2)	N/A	Y	PK	
	Ticket_No	numeric(10)	N/A	Y	FK	TICKET
	Line_Quantity	numeric(2)	N/A	Y		

Table 1: Data Dictionary for AMUSEMENT_PARK Database

1.1 Tasks

1.1.1 Creating Database

Create a database called AMUSEMENT_PARK and then select the database for use. After that, create the table structures for the database based on the ER model in Figure 1 and data directory in Table 1.

1.1.2 Loading Data

For each table structure, a file is provided as Comma Separated Values (CSV) format file. Every line in the CSV file contains row data with attributes aligned to that schema. You have to bulk insert the data in CSV files into the respective table structures using the following syntax.

```
BULK INSERT <table_name>
FROM 'path/to/csv_file'
WITH (
       FIELDTERMINATOR = ',' , -- CSV field delimiter
       ROWTERMINATOR = '0x0a' -- Shift the control to next row
)
```

You can access the data by right-clicking the icon and then selecting 'Save Embedded File to Disk...'

1.1.3 Querying Database

1. Write a query to display all Theme Parks except those in the UK.
2. Write a query to display the count of number of sales that occurred on the 26th November 2013.
3. Write a query to display the count of number of tickets for which price is between €20 and €22.
4. Display all attractions that have a capacity of more than 85 at the Theme Park FR1001.
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 ATTRACTION NO, HOUR RATE and the HOUR RATE with the 20% increase.
6. Write a query to count all the unique employees that exist in the HOURS table. Display the employee numbers of all employees and the total number of hours they have worked.
7. Show the attraction number and the minimum and maximum hourly rate for each attraction ordered by attraction number.
8. Display all information from the SALES table in descending order of the sale date.
9. Write a query to display the attraction number, employee name and the date they worked on the attraction. Order the results by the date worked.

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

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

12. Write a query which lists the approximate age of the employees on the company's tenth anniversary date (11/25/2016).

13. The Theme Park managers want to create a view called EMP_DETAILS which contains the following information. EMP_NO, PARK_CODE, PARK_NAME, EMP_NAME, EMP_HIRE_DATE and EMP_DOB.

14. Using your EMP_DETAILS view, write a query that displays the name of employees and the name of park they work at. 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 their name in lower case. Label the column USER_PASSWORD. You should also display the name and phone number of employee.

15. 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 2017.

2 MongoDB

In this exercise, you will create a small database and later query it to retrieve useful information. This will involve you creating a collection called RESTAURANTS in MongoDB. In order to interact with the MongoDB server, a client is necessary. The client will connect to the server and it will send the commands through the connection. You can use the Robo 3T[1] or MongoDB Compass[2] client.

```
{
    "address" : {
        "building" : "1007",
        "coord" : [ -73.856077 , 40.848447 ],
        "street" : "Morris Park Ave",
        "zipcode" : "10462"
    },
    "borough" : "Bronx",
    "cuisine" : "Bakery",
    "grades" : [
        {"date" : { "$date":1393804800000 }, "grade" : "A", "score" : 2 },
        {"date" : { "$date":1378857600000 }, "grade" : "A" , "score" : 6 },
        {"date" : { "$date":1358985600000 } , "grade" : "A" , "score" : 10 },
        {"date" : { "$date":1322006400000 } , "grade" : "A" , "score" : 9 },
        {"date" : { "$date":1299715200000 } , "grade" : "B" , "score" : 14 }
    ],
    "name" : "Morris Park Bake Shop",
    "restaurant_id" : "30075445"
}
```

Listing 1: Structure of RESTURANTS Collection

2.1 Tasks

2.1.1 Creating Database

Create a database called RESTAURANTS and then select the database for use. After that, create the collection called RESTAURANTS.

2.1.2 Loading Data

For the RESTAURANTS collection, a JSON file is provided. Every line in the JSON file contains a document. You have to bulk insert the data in the JSON file into the RESTAURANTS collection. You can take help from the source at: `https://stackoverflow.com/questions/19441228/insert-json-file-into-mongodb`.

You can access the data by right-clicking the icon and then selecting 'Save Embedded File to Disk...'

[1] `https://robomongo.org/`
[2] `https://www.mongodb.com/products/compass`

2.1.3 Querying Database

Write MongoDB queries to:

1. display the fields restaurant id, name, borough and cuisine for all the documents in the collection.
2. display the fields restaurant id, name, borough and zip code, but exclude the field id for all the documents in the collection.
3. display the first 5 restaurant which is in the borough Bronx.
4. find the restaurants who achieved a score more than 90.
5. find the restaurants that achieved a score, more than 80 but less than 100.
6. find the restaurants which belong to the borough Bronx and prepare either 'American' or 'Chinese' dish.
7. find the restaurant id, name, borough and cuisine for those restaurants that prepare dish except 'American' and 'Chinese'.
8. find the restaurant Id, name, and grades for those restaurants which achieved a grade of "A" and scored 11 on an ISODate "2014-08-11T00:00:00Z".
9. find the restaurants which do not prepare any cuisine of 'American' and achieved a score more than 70 and located in the longitude less than -65.754168.
10. find the restaurants which do not prepare any cuisine of 'American' and achieved a grade point "A" and do not belong to the borough 'Brooklyn'.

3 JDBC

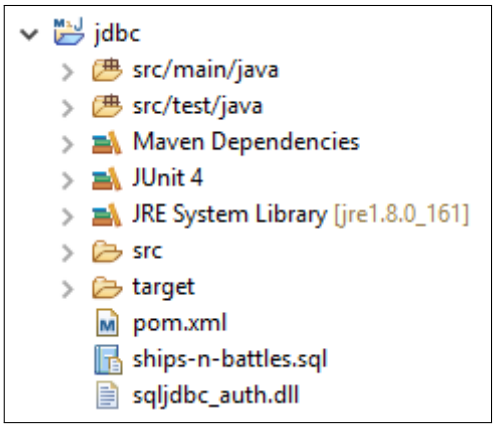
In this exercise, you will get familiar with the JDBC interface for connecting Java programs with SQL-based databases. You will learn how to use JDBC to connect to a relational database, how to execute SQL queries on the database, and how to handle the results of a query.

3.1 Getting Started

You can access the project files by right-clicking the icon and then selecting 'Save Embedded File to Disk...'

Next, import the project into Eclipse workspace, as noted here: `https://javapapers.com/java/import-maven-project-into-eclipse/`

After successfully importing the project into Eclipse workspace, you should have the following files and folders under the project directory:



3.1.1 ships-n-battles.sql

`ships-n-battles.sql` file contains the DDL and DML statements based on the following database schema:

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, startdate, enddate)
Outcomes(ship, battle, result)
```

The database and the table structures are created and the data is inserted into the database by the unit test called `setUpDatabase()` before any test is run from the `src/test/java/jdbc/JDBC_Test.java` file.

3.1.2 pom.xml

`pom.xml` contains information about the project and configuration details used by Maven to build the project. The cornerstone of this file is its dependency list. Maven allows users

to import dependencies into their software projects by adding dependencies to the `pom.xml` file. Once dependencies are added, they are automatically downloaded and updated.

3.2 Tasks

3.2.1 Change `serverName` and `instanceName`

Given below is the snapshot of the `src/test/java/jdbc/JDBC_Test.java` file. Open this file and change the `serverName` to the address of the server and `instanceName` to the instance to connect to on the server.

```
package jdbc;

import java.io.BufferedReader;

/**
 * Unit tests for JDBC.
 */

@RunWith(JUnit4.class)
public class JDBC_Test extends TestCase {

    static final String serverName = "WINDOWS-FUMCC90";
    static final String instanceName = "SQLEXPRESS";
    static Connection connection = null;

    public static void setUpDatabase() throws IOException, SQLException {

    @Test
    public void TestNumShipsInClass() throws SQLException {
        int NumShipsInClass = new JDBC().NumShipsInClass(connection, "Revenge");
        assertEquals(5, NumShipsInClass);
    }

    public void TestNumShipsSunkInBattle() throws SQLException {

    public void TestShipYearLaunched() throws SQLException {

    public void TestBattleFoughtBetweenDates() throws SQLException {

    public void TestGetAllBattles() throws SQLException {

    public static void cleanup() throws SQLException {

    public static <String> boolean listEqualsIgnoreOrder(List<String> list1, List<String> list2) {

}
```

3.2.2 Pass JUnit Tests

In this task, you will implement the methods in `src/main/java/jdbc/JDBC.java` file in order to pass the JUnit tests implemented in `src/test/java/jdbc/JDBC_Test.java` file.

For help, you can have a look at the JDBC Tutorial at: `https://www.tutorialspoint.com/jdbc/jdbc_tutorial.pdf`.

3.2.2.1 `TestNumShipsInClass()`

Provide an implementation of the `NumShipsInClass()` method in the `src/main/java/jdbc/JDBC.java` file that returns the number of ships belonging to a class given as an argument to the method.

3.2.2.2 `TestNumShipsSunkInBattle()`

Provide an implementation of the `NumShipsSunkInBattle()` method in the `src/main/java/jdbc/JDBC.java` file that returns the number of ships belonging to a class given as an argument to the method.

3.2.2.3 `TestShipYearLaunched()`

Provide an implementation of the `ShipYearLaunched()` method in the `src/main/java/jdbc/JDBC.java` file that returns the number of ships belonging to a class given as an argument to the method.

3.2.2.4 `TestBattleFoughtBetweenDates()`

Provide an implementation of the `BattleFoughtBetweenDates()` method in the `src/main/java/jdbc/JDBC.java` file that returns the number of ships belonging to a class given as an argument to the method.

3.2.2.5 `TestGetAllBattles()`

Provide an implementation of the `GetAllBattles()` method in the `src/main/java/jdbc/JDBC.java` file that returns the number of ships belonging to a class given as an argument to the method.