CHAPTER 3

Relational Database Management System: Oracle™

This chapter introduces the student to the basic utilities used to interact with Oracle DBMS. The chapter also introduces the student to programming applications in Java and JDBC that interact with Oracle databases. The discussion in this chapter is not specific to any version of Oracle and all examples would work with any version of Oracle higher than Oracle 8.

The company database of the Elmasri/Navathe text is used throughout this chapter. In Section 3.1 a larger data set is introduced for the company database. In Section 3.2 and 3.3, the SQL*Plus and the SQL*Loader utilities are introduced using which the student can create database tables and load the tables with data. Finally, in Section 3.4, programming in Java using the JDBC API and connecting to Oracle databases is introduced through three complete application programs.

3.1 COMPANY Database

Consider the COMPANY database state shown in Figure 5.6 of the Elmasri/Navathe text. Let us assume that the company is expanding with 3 new departments: Software, Hardware, and Sales. The Software department has 2 locations: Atlanta and Sacramento, the Hardware department is located in Milwaukee, and the Sales department has 5 locations: Chicago, Dallas, Philadelphia, Seattle, and Miami.

The Software department has 3 new projects: OperatingSystems, DatabaseSystems, and Middleware and the Hardware department has 2 new projects: InkjetPrinters and LaserPrinters. The company has added 32 new employees in this expansion process.

The updated `COMPANY` database is shown in the following tables.

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	22-May-78
	Administration	4	987654321	1-Jan-85
	Headquarters	1	888665555	19-Jun-71
	Software	6	111111100	15-May-99
	Hardware	7	444444400	15-May-98
	Sales	8	555555500	1-Jan-97

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4
	OperatingSystems	61	Jacksonville	6
	DatabaseSystems	62	Birmingham	6
	Middleware	63	Jackson	6
	InkjetPrinters	91	Phoenix	7
	LaserPrinters	92	LasVegas	7

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston
	6	Atlanta
	6	Sacramento
	7	Milwaukee
	8	Chicago
	8	Dallas
	8	Philadephia
	8	Seattle

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATION
	333445555	Alice	F	5-Apr-76	Daughter
	333445555	Theodore	M	25-Oct-73	Son
	333445555	Joy	F	3-May-48	Spouse
	987654321	Abner	M	29-Feb-32	Spouse
	123456789	Michael	M	1-Jan-78	Son
	123456789	Alice	F	31-Dec-78	Daughter
	123456789	Elizabeth	F	5-May-57	Spouse
	444444400	Johnny	M	4-Apr-97	Son
	444444400	Tommy	M	7-Jun-99	Son
	444444401	Chris	M	19-Apr-69	Spouse
	444444402	Sam	M	14-Feb-64	Spouse

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40
	453453453	1	20
	453453453	2	20
	333445555	2	10
	333445555	3	10
	333445555	10	10
	333445555	20	10
	999887777	30	30
	999887777	10	10
	987987987	10	35
	987987987	30	5
	987654321	30	20
	987654321	20	15
	888665555	20	null
	111111100	61	40
	111111101	61	40
	111111102	61	40
	111111103	61	40
	222222200	62	40
	222222201	62	48
	222222202	62	40
	222222203	62	40
	222222204	62	40
	222222205	62	40
	333333300	63	40
	333333301	63	46
	444444400	91	40
	444444401	91	40
	444444402	91	40
	444444403	91	40
	555555500	92	40
	555555501	92	44
	666666601	91	40
	666666603	91	40
	666666604	91	40
	666666605	92	40
	666666606	91	40
	666666607	61	40
	666666608	62	40
	666666609	63	40
	666666610	61	40
	666666611	61	40
	666666612	61	40
	666666613	61	30
	666666613	62	10
	666666613	63	10

EMPLOYEE									
FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DN
James	E	Borg	888665555	10-Nov-27	450 Stone, Houston, TX	M	55000	null	
Franklin	T	Wong	333445555	8-Dec-45	638 Voss, Houston, TX	M	40000	888665555	
Jennifer	S	Wallace	987654321	20-Jun-31	291 Berry, Bellaire, TX	F	43000	888665555	
John	B	Smith	123456789	9-Jan-55	731 Fondren, Houston, TX	M	30000	333445555	
Alicia	J	Zelaya	999887777	19-Jul-58	3321 Castle, Spring, TX	F	25000	987654321	
Ramesh	K	Narayan	666884444	15-Sep-52	971 Fire Oak, Humble, TX	M	38000	333445555	
Joyce	A	English	453453453	31-Jul-62	5631 Rice, Houston, TX	F	25000	333445555	
Ahmad	V	Jabbar	987987987	29-Mar-59	980 Dallas, Houston, TX	M	25000	987654321	
Jared	D	James	111111100	10-Oct-66	123 Peachtree, Atlanta, GA	M	85000	null	
Alex	D	Freed	444444400	9-Oct-50	4333 Pillsbury, Milwaukee, WI	M	89000	null	
John	C	James	555555500	30-Jun-75	7676 Bloomington, Sacramento, (	M	81000	null	
Jon	C	Jones	111111101	14-Nov-67	111 Allgood, Atlanta, GA	M	45000	111111100	
Justin	null	Mark	111111102	12-Jan-66	2342 May, Atlanta, GA	M	40000	111111100	
Brad	C	Knight	111111103	13-Feb-68	176 Main St., Atlanta, GA	M	44000	111111100	
Evan	E	Wallis	222222200	16-Jan-58	134 Pelham, Milwaukee, WI	M	92000	null	
Josh	U	Zell	222222201	22-May-54	266 McGrady, Milwaukee, WI	M	56000	222222200	
Andy	C	Vile	222222202	21-Jun-44	1967 Jordan, Milwaukee, WI	M	53000	222222200	
Tom	G	Brand	222222203	16-Dec-66	112 Third St, Milwaukee, WI	M	62500	222222200	
Jenny	F	Vos	222222204	11-Nov-67	263 Mayberry, Milwaukee, WI	F	61000	222222201	
Chris	A	Carter	222222205	21-Mar-60	565 Jordan, Milwaukee, WI	F	43000	222222201	
Kim	C	Grace	333333300	23-Oct-70	6677 Mills Ave, Sacramento, CA	F	79000	null	
Jeff	H	Chase	333333301	7-Jan-70	145 Bradbury, Sacramento, CA	M	44000	333333300	
Bonnie	S	Bays	444444401	19-Jun-56	111 Hollow, Milwaukee, WI	F	70000	444444400	
Alec	C	Best	444444402	18-Jun-66	233 Solid, Milwaukee, WI	M	60000	444444400	
Sam	S	Snedder	444444403	31-Jul-77	987 Windy St, Milwaukee, WI	M	48000	444444400	
Nandita	K	Ball	555555501	16-Apr-69	222 Howard, Sacramento, CA	M	62000	555555500	
Bob	B	Bender	666666600	17-Apr-68	8794 Garfield, Chicago, IL	M	96000	null	
Jill	J	Jarvis	666666601	14-Jan-66	6234 Lincoln, Chicago, IL	F	36000	666666600	
Kate	W	King	666666602	16-Apr-66	1976 Boone Trace, Chicago, IL	F	44000	666666600	
Lyle	G	Leslie	666666603	9-Jun-63	417 Hancock Ave, Chicago, IL	M	41000	666666601	
Billie	J	King	666666604	1-Jan-60	556 Washington, Chicago, IL	F	38000	666666603	
Jon	A	Kramer	666666605	22-Aug-64	1988 Windy Creek, Seattle, WA	M	41500	666666603	
Ray	H	King	666666606	16-Aug-49	213 Delk Road, Seattle, WA	M	44500	666666604	
Gerald	D	Small	666666607	15-May-62	122 Ball Street, Dallas, TX	M	29000	666666602	
Arnold	A	Head	666666608	19-May-67	233 Spring St, Dallas, TX	M	33000	666666602	
Helga	C	Pataki	666666609	11-Mar-69	101 Holyoke St, Dallas, TX	F	32000	666666602	
Naveen	B	Drew	666666610	23-May-70	198 Elm St, Philadelphia, PA	M	34000	666666607	
Carl	E	Reedy	666666611	21-Jun-77	213 Ball St, Philadelphia, PA	M	32000	666666610	
Sammy	G	Hall	666666612	11-Jan-70	433 Main Street, Miami, FL	M	37000	666666611	

3.2 *SQL*Plus* Utility

Oracle provides a command line utility, called SQL*Plus, that allows the user to execute SQL statements interactively. The user may enter the statements directly on the SQL*Plus prompt or may save the statements in a file and have them executed by entering the "start" command. We shall now look at how one can use this utility to create the database tables for the COMPANY database.

Creating Database Tables

Let us assume that all the SQL statements to create the COMPANY database tables are present in a file called company_schema.sql. The part that defines the DEPARTMENT table is shown below:

```
CREATE TABLE department (
  dname         varchar(25),
  dnumber       number(4),
  mgrssn        char(9),
  mgrstartdate date,
  primary key (dnumber),
  foreign key (mgrssn) references employee
);
```

The following SQL*Plus session illustrates how these statements would be executed resulting in the creation of the COMPANY database tables:

```
$ sqlplus
SQL>start company_schema

Table created

SQL>exit
$
```

The $ symbol used above is the command line prompt on a Unix system or a Windows command line interface.

In general, when several tables are defined in a single file, these definitions should be arranged in a particular order of foreign key references. For example, the definition of the WORKS_ON table should follow the definition of the EMPLOYEE table and the PROJECT table since it has foreign keys referencing the EMPLOYEE and the PROJECT tables.

Sometimes, it is possible for two tables to have a circular reference between them. For example, the foreign key constraints defined for the EMPLOYEE and the DEPARTMENT tables indicate a mutual dependency between the two tables. The EMPLOYEE table contains an attribute DNO which

refers to the DNUMBER attribute of the DEPARTMENT table and the MGRSSN attribute of the DEPARTMENT table refers to the SSN attribute of the EMPLOYEE table. This mutual dependency causes some trouble while creating the tables as well as loading the data. To avoid these problems, it may be simplest to omit the forward references, i.e. omit the DNO foreign key in the EMPLOYEE table.

A more appropriate way to handle the circular reference problem is as follows: Create the tables by omitting the forward references. After the tables are created, use the ALTER TABLE statement to add the omitted reference. In the case of the EMPLOYEE table, use the following ALTER command to add the forward reference after the DEPARTMENT table is created:

```
ALTER TABLE employee ADD (
  foreign key (dno) references department(dnumber)
);
```

The SQL*Plus utility is very versatile program and allows the user to submit any SQL statement for execution. In addition, it allows users to format query results in different ways. Please consult the SQL*Plus documentation that comes with any Oracle installation for further details on its use.

3.3 *SQL*Loader* Utility

Oracle provides a data loading utility called SQL*Loader[2] (sqlldr) that allows the user to populate the tables with data. To use sqlldr, we need a control file that indicates how that data is to be loaded and a data file that contains the data to be loaded. A typical control file is shown below:

```
LOAD DATA
INFILE <dataFile>
APPEND INTO TABLE <tableName>
FIELDS TERMINATED BY '<separator>'
(<attribute-list>)
```

Here, <datafile> is the name of the file containing the data, <tableName> is the database table into which the data is to be loaded, <separator> is a string that does not appear as part of the data, and <attribute-list> is the list of attributes or columns under which data is to be loaded. The attribute list need not be the entire list of columns and the attributes need not appear in the order in which they were defined in the create table statement,

As a concrete example, the following control file (department.ctl) will load data into the department table:

[2] For more details on this utility, please refer to Oracle's online documentation.

```
LOAD DATA
INFILE department.csv
APPEND INTO TABLE department
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(dname,dnumber,mgrssn,mgrstartdate DATE 'yyyy-mm-dd')
```

The `OPTIONALLY ENCLOSED` phrase is useful in situations when one of the data fields contained the separator symbol. In this case, the entire data value would be enclosed with a pair of double quotes. The `TRAILING NULLCOLS` phrase is useful in case there is a `null` value for the last data field. `Null` values are indicated by an empty string followed by the separator symbol. The corresponding data file, `department.csv` is:

```
Research,5,333445555,1978-05-22
Administration,4,987654321,1985-01-01
Headquarters,1,888665555,1971-06-19
Software,6,111111100,1999-05-15
Hardware,7,444444400,1998-05-15
Sales,8,555555500,1997-01-01
```

The following command will load the data into the department table:

```
$ sqlldr OracleId/OraclePassword control=department.ctl
```

Here, `OracleId/OraclePassword` is the Oracle user id/password. Upon execution of this command, a log file under the name `department.log` is created by `sqlldr`. This log file contains information such as the number of rows successfully loaded into the table, and any errors encountered while loading the data.

The loading of the data can be verified by the following `sqlplus` session:

```
$ sqlplus OracleId

SQL*Plus: Release 9.2.0.1.0 - Production on Wed Apr 6 20:35:45
2005

Copyright (c) 1982, 2002, Oracle Corporation.  All rights
reserved.

Enter password:

Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - 64bit Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
```

```
SQL> select * from department;

DNAME                          DNUMBER MGRSSN    MGRSTARTD
------------------------- ---------- --------- ---------
Research                            5 333445555 22-MAY-78
Administration                      4 987654321 01-JAN-85
Headquarters                        1 888665555 19-JUN-71
Software                            6 111111100 15-MAY-99
Hardware                            7 444444400 15-MAY-98
Sales                               8 555555500 01-JAN-97

6 rows selected.

SQL> exit
Disconnected from Oracle9i Enterprise Edition Release 9.2.0.1.0 -
64bit Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 – Production
$
```

It is possible to load the data without a separate data file by using the following control file:

```
LOAD DATA
INFILE *
APPEND INTO TABLE department
FIELDS TERMINATED BY ','
(dname,dnumber,mgrssn,mgrstartdate DATE 'yyyy-mm-dd')
BEGINDATA
Research,5,333445555,1978-05-22
Administration,4,987654321,1985-01-01
Headquarters,1,888665555,1971-06-19
Software,6,111111100,1999-05-15
Hardware,7,444444400,1998-05-15
Sales,8,555555500,1997-01-01
```

Note that the filename has been replaced by a "`*`" in the `INFILE` clause of the control file and the data is separated from the control information by the line containing the string `BEGINDATA`.

In the case of foreign key dependencies between tables, data for the tables that are being referenced should be loaded/created first before the data for the tables that refer to values in other tables. For example, the `EMPLOYEE` and `PROJECT` table data should be loaded before the data for the `WORKS_ON` table is loaded.

The circular references are trickier to handle as far as bulk loading is concerned. To successfully load such data, we may use NULL values for forward references and once the referenced data is loaded, we may use the UPDATE statement to add values for the forward references.

3.4 Programming with Oracle using the JDBC API

This section presents Oracle JDBC programming through three complete applications. The first example illustrates basic query processing via PreparedStatement object in Java. The query result has at most one answer. The second example illustrates basic query processing using Statement object. In this example, the query may have more than one answer. The final example is a more involved one in which recursive query as well as aggregate query processing is discussed.

To be able to program with Oracle databases using Java/JDBC, one must have access to an Oracle database installation with a listener program for JDBC connections. One must also have access to the JDBC drivers (classes12.zip or classes12.jar) that comes with any standard Oracle distribution. The driver archive file must be included in the Java CLASSPATH variable. Any standard Java environment is sufficient to compile and run these programs.

The COMPANY database of the Elmasri/Navathe text is used in each of the three examples that follow.

Example 1: A simple Java/JDBC program that prints the last name and salary of an employee given his or her social security number is shown below:

```
import java.sql.*;
import java.io.*;

class getEmpInfo {
  public static void main (String args [])
      throws SQLException, IOException {

    // Load Oracle's JDBC Driver
    try {
      Class.forName ("oracle.jdbc.driver.OracleDriver");
    } catch (ClassNotFoundException e) {
        System.out.println ("Could not load the driver");
      }

    //Connect to the database
    String user, pass;
    user = readEntry("userid  : ");
    pass = readEntry("password: ");
    Connection conn =
```

```
      DriverManager.getConnection
        ("jdbc:oracle:thin:@tinman.cs.gsu.edu:1521:sid9ir2",
         user,pass);

    //Perform query using PreparedStatement object
    //by providing SSN at run time
    String query =
      "select LNAME,SALARY from EMPLOYEE where SSN = ?";
    PreparedStatement p = conn.prepareStatement (query);
    String ssn = readEntry("Enter a Social Security Number: ");
    p.clearParameters();
    p.setString(1,ssn);
    ResultSet r = p.executeQuery();

    //Process the ResultSet
    if (r.next ()) {
      String lname = r.getString(1);
      double salary = r.getDouble(2);
      System.out.println(lname + "  " + salary);
    }

    //Close objects
    p.close();
    conn.close();
  }

  //readEntry function -- to read input string
  static String readEntry(String prompt) {
     try {
       StringBuffer buffer = new StringBuffer();
       System.out.print(prompt);
       System.out.flush();
       int c = System.in.read();
       while(c != '\n' && c != -1) {
         buffer.append((char)c);
         c = System.in.read();
       }
       return buffer.toString().trim();
     } catch (IOException e) {
       return "";
       }
   }

}
```

In the above program, the user is prompted for a database user id and password. The program uses this information to connect[3] to the database. The user is then prompted for the social security number of an employee. The program supplies this social security number as a parameter to an SQL query used by a `PreparedStatement` object. The query is executed and the resulting `ResultSet` is then processed and the last name and salary of the employee is printed to the console.

The JDBC API is a very simple API to learn as there are an few important classes and methods that are required to manipulate relational data. After the driver is loaded and a connection is made (quite standard code to do this), queries and updates are submitted to the database via one of three objects: `Statement`, `PreparedStatement`, and `CallableStatement`.

The `PreparedStatement` method is illustrated in Example 1 where the SQL statement is sent to the database server with possible place-holders for pluggable values for compilation using the `prepareStatement` method of the `Connection` object. After this is done, the same SQL statement may be executed a number of times with values provided for the place holder variables using the `setString` or similar methods. In Example 1, the prepared statement is executed only once.

The `Statement` method is more commonly used and is illustrated in the following example.

Example 2: Given a department number, the following Java/JDBC program prints the last name and salary of all employees working for the department.

```
import java.sql.*;
import java.io.*;

class printDepartmentEmps {
  public static void main (String args [])
      throws SQLException, IOException {

    //Load Oracle's JDBC Driver
    try {
      Class.forName ("oracle.jdbc.driver.OracleDriver");
    } catch (ClassNotFoundException e) {
        System.out.println ("Could not load the driver");
      }

    //Connect to the database
    String user, pass;
    user = readEntry("userid  : ");
    pass = readEntry("password: ");
    Connection conn =
    DriverManager.getConnection
```

[3] The connect string will be different in different installations of Oracle. Please consult your instructor/administrator for the exact connect string used in `getConnection` method.

```
        ("jdbc:oracle:thin:@tinman.cs.gsu.edu:1521:sid9ir2",
         user,pass);

    //Perform query using Statement object
    String dno = readEntry("Enter a Department Number: ");
    String query =
      "select LNAME,SALARY from EMPLOYEE where DNO = " + dno;
    Statement s = conn.createStatement();
    ResultSet r = s.executeQuery(query);

    //Process ResultSet
    while (r.next ()) {
      String lname = r.getString(1);
      double salary = r.getDouble(2);
      System.out.println(lname + "  " + salary);
    }

    //Close objects
    s.close();
    conn.close();
  }
```

The query is executed via a `Statement` object by including the department number as part of the query string at run time (using string concatenation). This is in contrast to the `PreparedStatement` method used in Example 1. The main difference between the two methods is that in the `PreparedStatement` method, the query string is first sent to the database server for syntax checking and then for execution subsequently. This method may be useful when the same query string is executed a number of times in a program with only a different parameter each time. On the other hand, in the `Statement` method, syntax checking and execution of the query happens at the same time.

The third method for executing SQL statements is to use the CallableStatement object. This is useful only in situations where the Java program needs to call a stored procedure or function. To learn about this method, the reader is referred to any standard JDBC textbook or the Oracle 9i Programming: A Primer (2004) published by Addison Wesley.

Example 3: In this next program, the user is presented with a menu of 3 choices:

(a) **Find supervisees at all levels:** In this option, the user is prompted for the last name of an employee. If there are several employees with the same last name, the user is presented with a list of social security numbers of employees with the same last name and asked to choose one. The program then proceeds to list all the supervisees of the employee and all levels below him or her in the employee hierarchy.

(b) **Find the top 5 highest paid employees**: In this option, the program finds five employees who rank in the top 5 in salary and lists them.

(c) **Find the top 5 highest worked employees:** In this option, the program finds five employees who rank in the top 5 in number of hours worked and lists them.

A sample interaction with the user is shown below:

```
$ java example3
Enter userid: book
Enter password: book

        QUERY OPTIONS
(a) Find Supervisees at all levels.
(b) Find Highest paid workers.
(c) Find the most worked workers.
(q) Quit.

Type in your option: a

Enter last name of employee : King
King,Kate 666666602
King,Billie 666666604
King,Ray 666666606
Select ssn from list : 666666602
     SUPERVISEES
FNAME          LNAME          SSN
---------------------------------------

Gerald         Small          666666607
Arnold         Head           666666608
Helga          Pataki         666666609
Naveen         Drew           666666610
Carl           Reedy          666666611
Sammy          Hall           666666612
Red            Bacher         666666613

        QUERY OPTIONS
(a) Find Supervisees at all levels.
(b) Find Highest paid workers.
(c) Find the most worked workers.
(q) Quit.

Type in your option: b

    HIGHEST PAID WORKERS
-------------------------------------------------

666666600 Bob            Bender          96000.0
222222200 Evan           Wallis          92000.0
444444400 Alex           Freed           89000.0
111111100 Jared          James           85000.0
```

```
555555500 John            James           81000.0

        QUERY OPTIONS
(a) Find Supervisees at all levels.
(b) Find Highest paid workers.
(c) Find the most worked workers.
(q) Quit.

Type in your option: c

    MOST WORKED WORKERS
-------------------------------------------------

666666613 Red             Bacher          50.0
222222201 Josh            Zell            48.0
333333301 Jeff            Chase           46.0
555555501 Nandita         Ball            44.0
111111100 Jared           James           40.0

        QUERY OPTIONS
(a) Find Supervisees at all levels.
(b) Find Highest paid workers.
(c) Find the most worked workers.
(q) Quit.

Type in your option: q

$
```

The program for option (a) is discussed now. Finding supervisees at all levels below an employee is a recursive query in which the data tree needs to be traversed from the employee node all the way down to the leaves in the sub-tree. One common strategy to solve this problem is to use a temporary table of social security numbers. Initially, this temporary table will store the next level supervisees. Subsequently, in an iterative manner, the supervisees at the "next" lower level will be computed with a query that involves the temporary table. These next supervisees are then added to the temporary table. This iteration is continued as long as there are new social security numbers added to the temporary table in any particular iteration. Finally, when no new social security numbers are found, the iteration is stopped and the names and social security numbers of all supervisees in the temporary table are listed.

The section of the main program that (a) creates the temporary table, (b) calls the `findSupervisees` method, and (c) drops the temporary table is shown below:

```
    /* create new temporary table called tempSSN */
    String sqlString = "create table tempSSN (" +
                       "ssn char(9) not null, " +
                       "primary key(ssn))";
```

```
   Statement stmt1 = conn.createStatement();
   try {
      stmt1.executeUpdate(sqlString);
   } catch (SQLException e) {
       System.out.println("Could not create tempSSN table");
       stmt1.close();
       return;
     }

…
…
       case 'a': findSupervisees(conn);
…
…
   /* drop table called tmpSSN */
   sqlString = "drop table tempSSN";
   try {
      stmt1.executeUpdate(sqlString);
   } catch (SQLException e) {
     }
```

The `findSupervisees` method is divided into four stages.

Stage 1: The program prompts the user for input data (last name of employee) and queries the database for the social security number of employee. If there are several employees with same last name, the program lists all their social security numbers and prompts the user to choose one. At the end of this stage, the program would have the social security number of the employee whose supervisees the program needs to list. The code for this stage is shown below.

```
private static void findSupervisees(Connection conn)
     throws SQLException, IOException {

   String sqlString = null;

   Statement stmt = conn.createStatement();

   // Delete tuples from tempSSN from previous request.
   sqlString = "delete from tempSSN";
   try {
     stmt.executeUpdate(sqlString);
   } catch (SQLException e) {
       System.out.println("Could not execute Delete");
       stmt.close();
       return;
     }

   /* Get the ssn for the employee */
```

```
sqlString = "select lname, fname, ssn " +
            "from   employee " +
            "where  lname = '";
String lname =
   readEntry( "Enter last name of employee : ").trim();
sqlString += lname;
sqlString += "'";
ResultSet rset1;
try {
  rset1 = stmt.executeQuery(sqlString);
} catch (SQLException e) {
    System.out.println("Could not execute Query");
    stmt.close();
    return;
}
String samelName[] = new String[40];
String fName[] = new String[40];
String empssn[] = new String[40];
String ssn;
int nNames = 0;
while (rset1.next()) {
  samelName[nNames] = rset1.getString(1);
  fName[nNames] = rset1.getString(2);
  empssn[nNames] = rset1.getString(3);
  nNames++;
}
if (nNames == 0) {
  System.out.println("Name does not exist in database.");
  stmt.close();
  return;
}
else if (nNames > 1) {
  for(int i = 0; i < nNames; i++) {
    System.out.println(samelName[i] + "," +
                       fName[i] + " " + empssn[i]);
  }
  ssn = readEntry("Select ssn from list : ");
  ResultSet r = stmt.executeQuery(
       "select ssn from employee where ssn = '" + ssn + "'");
  if( !r.next()) {
    System.out.println("SSN does not exist in database.");
    stmt.close();
    return;
  }
}
else {
  ssn = empssn[0];
```

```
}
```

Stage 2: In the second stage, the `findSupervisees` method finds the immediate supervisees, i.e. supervisees who directly report to the employee. The social security numbers of immediate supervisees are then inserted into the temporary table.

```
/* Find immediate supervisees for that employee */
sqlString =
  "select distinct ssn from employee where superssn = '";
sqlString += ssn;
sqlString += "'";
try {
  rset1 = stmt.executeQuery(sqlString);
} catch (SQLException e) {
    System.out.println("Could not execute query");
    stmt.close();
    return;
}

/* Insert result into tempSSN table*/
Statement stmt1 = conn.createStatement();
while (rset1.next()) {
  String sqlString2 = "insert into tempSSN values ('";
  sqlString2 += rset1.getString(1);
  sqlString2 += "')";
  try {
    stmt1.executeUpdate(sqlString2);
  } catch (SQLException e) {
   }
}
```

Stage 3: In the third stage, the `findSupervisees` method iteratively calculates supervisees at the next lower level using the query:

```
select employee.ssn
from   employee, tempSSN
where  superssn = tempSSN.ssm;
```

The results of this query are inserted back into the `tempSSN` table to prepare for the next iteration. A boolean variable, called `newrowsadded`, is used to note if any new tuples were added to the `tempSSN` table in any particular iteration. Since the `tempSSN` table's only column (`ssn`) is also defined as a primary key, duplicate social security numbers would cause an SQLException which is simply ignored in this program. The code for this stage is shown below:

```
/* Recursive Querying */
ResultSet rset2;
```

```
boolean newrowsadded;
sqlString = "select employee.ssn from employee, tempSSN " +
            "where superssn = tempSSN.ssn";
do {
  newrowsadded = false;
  try {
    rset2 = stmt.executeQuery(sqlString);
  } catch (SQLException e) {
      System.out.println("Could not execute Query");
      stmt.close();
      stmt1.close();
      return;
  }
  while ( rset2.next()) {
    try {
      String sqlString2 = "insert into tempSSN values ('";
      sqlString2 += rset2.getString(1);
      sqlString2 += "')";
      stmt1.executeUpdate(sqlString2);
      newrowsadded = true;
    } catch (SQLException e) {
      }
  }
} while (newrowsadded);
stmt1.close();
```

Stage 4: In the final stage, the `findSupervisees` method prints names and social security numbers of all employees whose social security number is present in the `tempSSN` table. The code is shown below:

```
/* Print Results */
sqlString = "select fname, lname, e.ssn from " +
            "employee e, tempSSN t where e.ssn = t.ssn";
ResultSet rset3;
try {
  rset3 = stmt.executeQuery(sqlString);
} catch (SQLException e) {
    System.out.println("Could not execute Query");
    stmt.close();
    return;
 }
 System.out.println("     SUPERVISEES ");
 System.out.print("FNAME");
 for (int i = 0; i < 10; i++)
   System.out.print(" ");
 System.out.print("LNAME");
```

```
    for (int i = 0; i < 10; i++)
      System.out.print(" ");
    System.out.print("SSN");
    for (int i = 0; i < 6; i++)
      System.out.print(" ");
    System.out.println("\n---------------------------------\n");

    while(rset3.next()) {
      System.out.print(rset3.getString(1));
      for (int i = 0;
           i < (15 - rset3.getString(1).length()); i++)
        System.out.print(" ");
      System.out.print(rset3.getString(2));
      for (int i = 0;
           i < (15 - rset3.getString(2).length()); i++)
        System.out.print(" ");
      System.out.println(rset3.getString(3));
    }
    stmt.close();
  }
```

This concludes the discussion of the `findSupervisees` method. The code to implement options (b) and (c) is quite straightforward and is omitted. The entire code for all the examples is available along with this lab manual.

Exercises

1. Consider the ER-schema generated for the `UNIVERSITY` database in Laboratory Exercise 7.31 of the Elmasri/Navathe text. Convert the schema to a relational database and solve the following:
    a. Create the database tables in Oracle using the `SQL*Plus` interface.
    b. Create comma separated data files containing data for at least 3 departments (CS, Math, and Biology), 20 students, and 20 courses. You may evenly distribute the students and the courses among the three departments. You may also choose to assign minors to some of the students. For a subset of the courses, create sections for the Fall 2006 and Spring 2007 terms. Make sure that you assign multiple sections for some courses. Assuming that the grades are available for Fall 2006 term, add enrollments of students in sections and assign numeric grades for these enrollments. Do not add any enrollments for the Spring 2007 sections.
    c. Using the `SQL*Loader` utility of Oracle, load the data created in (b) into the database.
    d. Write SQL queries for the following and execute them within an `SQL*Plus` session.
        i. Retrieve student number, first and last names, and major department names of students who have a minor in the Biology department.

ii. Retrieve the student number and the first and last names of students who have never taken a class with instructor named "King".
iii. Retrieve the student number and the first and last names of students who have taken classes only with instructor named "King".
iv. Retrieve department names of departments along with the numbers of students with that department as their major (sorted in decreasing order of the number of students).
v. Retrieve department names of departments along with the numbers of students with that department as their major (sorted in decreasing order of the number of students).
vi. Retrieve the instructor names of all instructors teaching CS courses along with the sections (course number, section, semester, and year) they are teaching and the total number of students in these sections.
vii. Retrieve the student number, first and last names, and major departments of students who do not have a grade of "A" in any of the courses they have enrolled in.
viii. Retrieve the student number, first and last names, and major departments of all straight-A students (students who have a grade of "A" in all of the courses they have enrolled in).
ix. For each student in the CS department, retrieve their student number, first and last names, and their GPA.

2. Consider the ER-schema generated for the `MAIL_ORDER` database in Laboratory Exercise 7.32 of the Elmasri/Navathe text. Convert the schema to a relational database and solve the following:
    a. Create the database tables in Oracle using the `SQL*Plus` interface.
    b. Create comma separated data files containing data for at least 6 customers, 6 employees, and 20 parts. Also create data for 20 orders with an average of 3 to 4 parts in each order.
    c. Using the `SQL*Loader` utility of Oracle, load the data created in (b) into the database.
    d. Write SQL queries for the following and execute them within an `SQL*Plus` session.
        i. Retrieve the names of customers who have ordered at least one part costing more than $30.00.
        ii. Retrieve the names of customers who have ordered all parts that cost less than $20.00.
        iii. Retrieve the names of customers who order parts only from employees who live in the same city as they live in.
        iv. Retrieve a list of part number, part name, and total quantity ordered for that part. Produce a listing sorted in decreasing order of the total quantity ordered.
        v. Retrieve the average waiting time (number of days) for all orders. The waiting time is defined as the difference between shipping date and order received date rounded up to the nearest day.

vi. For each employee, retrieve his number, name, and total sales (in terms of dollars) in a given year, say 2005.

3. Consider the relational schema for the GRADE_BOOK database in Laboratory Exercise 7.3 of the Elmasri/Navathe text augmented with the following two tables:

```
components(Term,Sec_no,Comp_name,Max_points,Weight)
scores(Sid,Term,Sec_no,Comp_name,Points)
```

where the components table records the grading components for a course and the scores table records the points earned by a student in a grading component of the course.

a. Create the database tables in Oracle using the SQL*Plus interface.
b. Create comma separated data files containing data for at least 20 students and 3 courses with an average of 8 students in each course. Create data for 3 to 4 grading components for each course. Then, create data to assign points to each student in every course they are enrolled in for each of the grading components.
c. Using the SQL*Loader utility of Oracle, load the data created in (b) into the database.
d. Write SQL queries for the following and execute them within an SQL*Plus session.
    i. Retrieve the names of students who have enrolled in 'CSc 226' or 'CSc 227'.
    ii. Retrieve the names of students who enrolled in some class during fall 2005 but no class in spring 2006.
    iii. Retrieve the titles of courses that have had enrollments of five or fewer students.
    iv. Retrieve the student ids, their names, and their course average (weighted average of points from all components) for the CSc 226 course offering with section number 2 during Fall 2005.

NOTE: All references to Exercises and Laboratory Exercises in the following problems refer to the numbering in the Elmasri/Navathe text.

4. Consider the UNIVERSITY database of Exercise 1. Write and test a Java program that performs the functions illustrated in the following terminal session:

```
$ java p1
Student Number: 1234
Semester: Fall
Year: 2005

Main Menu
            (1) Add a class
            (2) Drop a class
            (3) See my schedule
            (4) Exit
```

```
Enter your choice: 1

Course Number: CSC 1010
Section: 2
Class Added


Main Menu
            (1) Add a class
            (2) Drop a class
            (3) See my schedule
            (4) Exit
Enter your choice: 1

Course Number: MATH 2010
Section: 1
Class Added


Main Menu
            (1) Add a class
            (2) Drop a class
            (3) See my schedule
            (4) Exit
Enter your choice: 3

Your current schedule is:

CSC 1010 Section 2, Introduction to Computers, Instructor: Smith
MATH 2010 Section 1, Calculus I, Instructor: Jones


Main Menu
            (1) Add a class
            (2) Drop a class
            (3) See my schedule
            (4) Exit
Enter your choice: 2

Course Number: CSC 1010
Section: 2
Class dropped

Main Menu
            (1) Add a class
            (2) Drop a class
            (3) See my schedule
            (4) Exit
```

```
Enter your choice: 3

Your current schedule is:

MATH 2010 Section 1, Calculus I, Instructor: Jones


Main Menu
            1. Add a class
            2. Drop a class
            3. See my schedule
            4. Exit
Enter your choice: 4
$
```

As the terminal session shows, the student signs into this program with an id number and the term and year of registration. The `Add a class` option allows the student to add a class to his/her current schedule and the `Drop a class` option allows the student to drop a class in his/her current schedule. The `See my schedule` option lists the classes in which the student is registered.

5. Consider the `MAIL_ORDER` database of Exercise 2. Write and test a Java program that prints an invoice for a given order number as illustrated in the following terminal session:

```
$java p2
Order Number: 1020

Customer:      Charles Smith
Customer No:   1111
Zip:           67226

Order No:      1020
Taken By:      John Jones (emp. No. 1122)
Received on:   10-DEC-1994
Shipped on:    13-DEC-1994

Part No.  Part Name Quantity  Price     Sum
-------------------------------------------
10506     Nut       200       1.99      398.00
10509     Bolt      100       1.55      155.00
-------------------------------------------
                              Total:    553.00
```

6. Consider the GRADE_BOOK database of Exercise 3. Write and test a Java program that interacts with the Oracle database and prints a grade count report for each course. The report should list the number of A's, B's, C's, D's, and F's given in each of the courses sorted on course number. The format for the report is:

```
                    Grade Count Report
CNO       Course Title   #A's #B's #C's #D's #F's
-------------------------------------------------
CSc2010   Intro to CS    12   8    15   5    2
CSc2310   Java           15   2    12   8    1
-------------------------------------------------
```