Jackie Diep

CSC 411 DBMS Design

Assignment #4

4/2/2021

Chapter 7 [5 points for each]

7.23) Explain what is meant by repetition of information and inability to represent information. Explain why each of these properties may indicate a bad relational-database design.

Repetition of Information is when a relation has a value that is dependent on another value in the same relation, and they are repeated consistently throughout.

This is a problem because it can cause data inconsistency when changes are made, and it increases the amount of storage required.

Inability to represent information is when a relation only exists between a specific set of information.

This is a problem because information cannot be put into the relation without also including irrelevant information and null values.

7.24) Why are certain functional dependencies called *trivial* functional dependencies?

A trivial function dependency exists when a set of attributes used to describe an attribute include that attribute.
b → a if a is a subset of b

7.37) List the three design goals for relational databases, and explain why each is desirable.

1. BCNF (Boyce-Codd normal form)
    a. It eliminates all redundancy that can be discovered based on functional dependencies
2. Losslessness
    a. Information is not lost during decomposition, and the original relation can be restored using a join
3. Dependency Preservation

a. All attributes can be checked in one relation, which is faster and easier to manage than using joins

7.38) In designing a relational database, why might we choose a non-BCNF design?

Sometimes, a database that is both BCNF and dependency preserving is impossible, so if a dependency preserving database is preferable, then a different form may be chosen.

Chapter 7 [10 points for each]

7.21) Give a lossless decomposition into BCNF of schema R of Exercise 7.1.

From the dependencies given in 7.1,
- $B \rightarrow D$ is non-trivial
- $B \rightarrow ABCDE$ cannot be derived from F

s.t. (R-D)U(B,D)

= {(A, B, C, E),(B, D)}

7.22) Give a lossless, dependency-preserving decomposition into 3NF of schema R of Exercise 7.1.

$R_1 \cap R_2 = (A)$
$A^+ = R_1 + (R_2 - A) = (A, B, C, D, E)$, so it is lossless
All dependencies given in 7.1 in R_1 and R_2 can be covered from A^+, so it is dependency preserving

= {(A, B, C), (C, D, E), (B, D), (E, A)}

Chapter 5 [15 points for each]

5.12 - Write a Java program that allows university administrators to print the teaching record of an instructor.

*** Used Figure 5.1 in Database System Concepts 7th Ed. as reference

a)Start by having the user input the login ID and password; then open the proper connection.

```
// Scanner for input
import java.util.Scanner;

class Main {
public static void JavaInstSearch(String[] args) {

// Create scanner
        Scanner scanner = new Scanner(System.in);

// Prompt for user + pw
        System.out.println("Enter username");
        String userName = scanner.nextLine();

        System.out.println(“Enter password”);
        String pWord = scanner.nextLine();

// connect to db using user inputs
        try (
                Connection connect = DriverManager.getConnection(
                        “jdbc:mysql:@instExample.edu:12001:exampledb”,
                        userName, pWord);
                Statement exStatement = connect.createStatement();
        )
        catch (Exception sqle)
        {
                System.out.println(“Exception caught : “ + sqle);
        }}}
```

b)The user is asked next for a search substring and the system returns (ID, name) pairs of instructors whose names match the substring. Use the like ('%substring%') construct in SQL to do this. If the search comes back empty, allow continued searches until there is a nonempty result.

```
import java.util.Scanner;
class Main {
public static void JavaInstSearch(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter username");
        String userName = scanner.nextLine();

        System.out.println("Enter password");
        String pWord = scanner.nextLine();

// Prepare a string for input
        String searchName = "SELECT ID, name FROM instructor WHERE
                name LIKE ?";

        try (
                Connection connect = DriverManager.getConnection(
                        "jdbc:mysql:@instExample.edu:12001:exampledb",
                        userName, pWord);
                Statement exStatement = connect.createStatement();
        ) {

// create a prepared string object using the previously made string
                PreparedStatement updateString =
                        connect.prepareStatement(updateString);)

// make int check for do-while loop
                        int check = 0;
                        do
                        {

// prompt for instructor name
                                System.out.println("Enter instructor name");
                                String instName = scanner.nextLine();
```

```
// Update the string with %instName%
                        updateString.setString(1, "%" + instName + "%");
                        ResultSet r1 = exStatement.executeQuery();
// if no match was found, reset instName and go back through the loop
                        if(r1.next() == false)
                        {
                                String instName = "";
                                updateString.setString(1, instName);
                                System.out.println("No instructor found");
                        }
// if a match was found, display until the table's end and exit the loop
                        else
                        {
                                while(r1.next())
                                {
                                        System.out.println(r1.getInt(1) + "\t" +
                                                r1.getString("name");
                                }
                                check = 1;
                        }
                } while (check = 0);
        }
        catch (Exception sqle)
        {
                System.out.println("Exception caught : " + sqle);
        }
}
}
```

5.13) Suppose you were asked to define a class MetaDisplay in Java, containing a method static void printTable(String r); the method takes a relation name r as input, executes the query "select * from r", and prints the result out in tabular format, with the attribute names displayed in the header of the table.

a) **What do you need to know about relation r to be able to print the result in the specified tabular format?**

The names and amount of columns. While the amount of names will give you the amount, the amount will help if using code to find the names.

b) **What JDBC methods(s) can get you the required information?**

Two methods from the ResultSetMetaData java interface :

getColumnCount() and getColumnName()

c) **Write the method printTable(String r) using the JDBC API.**

```
// Scanner for input
import java.util.Scanner;
class Main {
public static void JavaInstSearch(String[] args) {
// Create scanner
        Scanner scanner = new Scanner(System.in);
// Prompt for user + pw
        System.out.println("Enter username");
        String userName = scanner.nextLine();
        System.out.println("Enter password");
        String pWord = scanner.nextLine();
// connect to db using user inputs
        try (
                Connection connect = DriverManager.getConnection(
                        "jdbc:mysql:@instExample.edu:12001:exampledb",
                        userName, pWord);
                Statement exStatement = connect.createStatement();)
```

```
        {


// Run the query and obtain the meta data
                String search = “select * from r”;
                ResultSet r1 = exStatement.executeQuery(search);
                ResultSetMetaData metaData1 = r1.getMetaData();

// Get the column count and assign it to an integer
                int colCount = metaData1.getColumnCount();

// Print the column names in a tabular format
                for(int i = 0; i < colCount; i++)
                {
                        System.out.print(metaData1.getColumnName(i) + “\t”);
                }

// Until the end of the table, print the values of the table in tabular format
                while(r1.next())
                {
                        for(int i = 0; i< colCount; i++)
                        {

// Left this statement blank, as I do not know the name of the columns. If there is a way to do this in code before running the getColumnName(i) method, then I could not figure it out.
                                System.out.println()
                        }

                }
        }
        catch (Exception sqle)
        {
                System.out.println(“Exception caught : “ + sqle);
        }}}
```