CHAPTER 9

Application Design and Development

Exercises

9.12 Write a servlet and associated HTML code for the following very simple application: A user is allowed to submit a form containing a value, say n, and should get a response containing n "*" symbols.

Answer:

HTML form

```
<html>
    <head>
        <title>DB Book Exercise 8.8 </title>
    </head>
    <form action="servlet/StarServlet" method=get>
        Enter the value for "n"
        <br>
        <input type=text size=5 name="n">
        <input type=submit value="submit">
    </form>
</html>
```

Servlet Code

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class StarServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        int n = Integer.parseInt(request.getParameter("n"));
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HEAD><TITLE>Exercise 8.8</TITLE></HEAD>");
        out.println("<BODY>");
        for (int i = 0; i < n; i++) {
            out.print("*");
        }
        out.println("</BODY>");
        out.close();
    }
}
```

9.13 Write a servlet and associated HTML code for the following simple application: A user is allowed to submit a form containing a number, say n, and should get a response saying how many times the value n has been submitted previously. The number of times each value has been submitted previously should be stored in a database.

Answer: **HTML form**

```
<html>
    <head>
        <title>DB Book Exercise 9.13 </title>
    </head>
    <form action="servlet/KeepCountServlet" method=get>
        Enter the value for "n"
        <br>
        <input type=text size=5 name="n">
        <input type=submit value="submit">
    </form>
</html>
```

Schema

```
CREATE TABLE SUBMISSION_COUNT (
    value integer unique,
    scount integer not null);
```

Servlet Code

```
import java.io.*; import java.sql.*; import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class KeepCountServlet extends HttpServlet {
    private static final String query =
        "SELECT scount FROM SUBMISSION_COUNT WHERE value=?";

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        int n = Integer.parseInt(request.getParameter("n"));
        int count = 0;
        try {
            Connection conn = getConnection();
            PreparedStatement pstmt = conn.prepareStatement(query);
            pstmt.setInt(1, n);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                count = rs.getInt(1);
            }
            pstmt.close();

            Statement stmt = conn.createStatement();
            if (count == 0) {
                stmt.executeUpdate("INSERT INTO SUBMISSION_COUNT VALUES ("
                        + n + ", 1)");
            } else {
                stmt.executeUpdate("UPDATE SUBMISSION_COUNT SET " +
                        "scount=scount+1 WHERE value=" + n);
            }
            stmt.close();
            conn.close();
        }
        catch(Exception e) {
            throw new ServletException(e.getMessage());
        }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HEAD><TITLE>Exercise 9.13</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("The value "+n+ " has been submitted "+count+" times previously.");
        out.println("</BODY>");
        out.close();
    }
}
```

9.14 Write a servlet that authenticates a user (based on user names and passwords stored in a database relation), and sets a session variable called *userid* after authentication.

Answer: **HTML form**

```
<html>
    <head>
        <title>DB Book Exercise 9.14 </title>
    </head>
    <form action="servlet/SimpleAuthServlet" method=get>
        User Name:
        <input type=text size=20 name="user">
        <BR>
        <BR>
        Password :
        <input type=password size=20 name="passwd">
        <BR>
        <input type=submit value="submit">
    </form>
</html>
```

Schema

```
CREATE TABLE USERAUTH(
    userid integer primary key,
    username varchar(100) unique,
    password varchar(20)
);
```

Servlet Code

```
import java.io.*; import java.sql.*; import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleAuthServlet extends HttpServlet {
    private static final String query =
            "SELECT userid, password FROM USERAUTH WHERE username=?";

    public void doGet(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException
    {
        String user = request.getParameter("user");
        String passwd = request.getParameter("passwd");
        String dbPass = null;
        int userId = -1;
        try {
            Connection conn = getConnection();
            PreparedStatement pstmt = conn.prepareStatement(query);
            pstmt.setString(1, user);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                userId = rs.getInt(1);
                dbPass = rs.getString(2);
            }
            pstmt.close();
            conn.close();
        }
        catch(Exception e) {
            throw new ServletException(e.getMessage());
        }
        String message;
        if(passwd.equals(dbPass)) {
            message = "Authentication successful";
            getServletContext().setAttribute("userid", new Integer(userId));
        } else {
            message = "Authentication failed! Please check the username " +
                "and password.";
        }

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HEAD><TITLE>Exercise 9.14</TITLE></HEAD>");
        out.println("<BODY>");
        out.println(message);
        out.println("</BODY>");
        out.close();
    }
}
```

9.15 What is an SQL injection attack? Explain how it works, and what precautions must be taken to prevent SQL injection attacks.

Answer:

SQL injection attack occurs when a malicious user (attacker) manages to get an application to execute an SQL query created by the attacker. If an application constructs an SQL query string by concatenating the user supplied parameters, the application is prone to SQL injection attacks.

For example, suppose an application constructs and executes a query to retrieve a user's password in the following way:

```
String userid = request.getParameter("userid");
executeQuery("SELECT password FROM userinfo WHERE userid= ' " + userid + " ' ");
```

Now, if a user types the value for the parameter as:

```
john' OR userid= 'admin
```

the query constructed will be:

```
SELECT password FROM userinfo WHERE userid='john' OR userid='admin';
```

This can reveal unauthorized information to the attacker.

Prevention:

Use prepared statements, with any value that is taken as user input (not just text fields, but even options in drop-down menus) passed as a parameter; user input should **never** be concatenated directly into a query string. The JDBC, ODBC, ADO.NET, or other libraries that provide prepared statements ensure that special characters like quotes are escaped as appropriate for the target database, so that SQL injection attempts will fail.

9.16 Write pseudocode to manage a connection pool. Your pseudocode must include a function to create a pool (providing a database connection string, database user name, and password as parameters), a function to request a connection from the pool, a connection to release a connection to the pool, and a function to close the connection pool.

Answer:

```
// This connection pool manager is NOT thread-safe.

INITIAL_POOL_SIZE = 20;
POOL_SIZE_INCREMENT = 5;
MAX_POOL_SIZE = 100;
Queue freeConnections = empty queue;
Queue activeConnections = empty queue;
String poolConnURL;
String poolConnUserId;
String poolConnPasswd;

createPool(connString, userid, password) {
      poolConnURL = connString;
      poolConnUserId = userid;
      poolConnPasswd = password;
      for (i = 0; i < INITIAL_POOL_SIZE; i++) {
            conn = createConnection(connString, userid, password);
            freeConnections.add(conn);
      }
}

Connection getConnection() {
      if(freeConnections.size() != 0){
            conn = freeConnections.remove();
            activeConnections.add(conn);
            return conn;
      }
      activeConns = activeConnections.size();
      if (activeConns == MAX_POOL_SIZE)
            ERROR("Max pool size reached");
      if (MAX_POOL_SIZE − activeConns > POOL_SIZE_INCREMENT)
            connsToCreate = POOL_SIZE_INCREMENT;
      else
            connsToCreate = MAX_POOL_SIZE − activeConns;

      for (i = 0; i < connsToCreate; i++) {
            conn = createConnection(poolConnURL, poolConnUserId,
                        poolConnPasswd);
            freeConnections.add(conn);
      }
      return getConnection();
}

releaseConnection(conn) {
      activeConnections.remove(conn);
      freeConnections.add(conn);
}
```

```
closePool() {
    if(activeConnections.size() != 0)
        WARNING("Connections active. Will force close.");
    for (i=0; i < freeConnections.size(); i++) {
        conn = freeConnections.elementAt(i);
        freeConnections.removeElementAt(i);
        conn.close();
    }

    for (i=0; i < activeConnections.size(); i++) {
        conn = activeConnections.elementAt(i);
        activeConnections.removeElementAt(i);
        conn.close();
    }
}
```

9.17 Explain the terms CRUD and REST.

Answer:

The term CRUD refers to simple user interfaces to a relation (or an object model), that provide the ability to Create tuples, Read tuples, Update tuples and Delete tuples (or objects).

In Representation State Transfer (or REST), Web service function calls are executed by a standard HTTP request to a URL at an application server, with parameters sent as standard HTTP request parameters. The application server executes the request (which may involve updating the database at the server), generates and encodes the result, and returns the result as the result of the HTTP request. The server can use any encoding for a particular requested URL; XML and the JavaScript Object Notation (JSON) encoding are widely used. The requestor parses the returned page to access the returned data.

9.18 Many Web sites today provide rich user-interfaces using Ajax. List two features each of which reveals if a site uses Ajax, without having to look at the source code. Using the above features, find three sites which use Ajax; you can view the HTML source of the page to check if the site is actually using Ajax.

Answer:

a. A Web site with a form that allows you to select a value from one menu (e.g. country), and once a value has been selected, you are allowed to select a value from another menu (e.g. state from a list of states in that country) with the list of values for the second menu (e.g. state) empty until the first value (e.g. country) is selected, probably uses Ajax. If the number of countries is small, the site may well send all country-state pairs ahead of time, and then simply use Javascript without Ajax; however, if you notice a small delay in populating the second menu, the site probably uses Ajax.

b. A Web site that supports autocompletion for text that you are typing almost surely uses Ajax. For example, a search Web site that suggests possible completions of your query as you type the query in, or a Web-based email site that suggests possible completions of an email address as you type in the address almost surely use Ajax to communicate with a server after you type in each character (sometimes starting after the 3rd or 4th character), and respond with possible completions.

c. A Web form that, on filling in one piece of data, such as your email address or employee code, fills in other fields such as your name and contact information, without refreshing the page, almost surely uses Ajax to retrieve required information using the information (such as the email address or employee code) provided by the user.

Popular Web sites that use Ajax include almost all current generation Web email interfaces (such as GMail, Yahoo! mail, or Windows Live mail), and almost all search engines, which provide autocompletion. Online document management systems such as Google Docs or Office Live use Ajax extensively to push your updates to the server, and to fetch concurrent updates (to different parts of the document or spreadsheet) transparently. Check your organizations Web applications to find more local examples.

9.19 XSS attacks:

a. What is an XSS attack?

b. How can the referer field be used to detect some XSS attacks?

Answer:

a. In an XSS attack, a malicious user enters code written in a client-side scripting language such as JavaScript or Flash instead of entering a valid name or comment. When a different user views the entered text, the browser would execute the script, which can carry out actions such as sending private cookie information back to the malicious user, or execute an action on a different Web site, such as a bank Web site, that the user may be logged into.

b. The HTTP protocol allows a server to check the **referer** of a page access, that is, the URL of the page that had the link that the user clicked on to initiate the page access. By checking that the referer is valid, for example, that the referer URL is a page on the same Web site (say the bank Web site in the previous example), XSS attacks that originated on a different Web site accessed by the user can be prevented. The referer field is set by the browser, so a malicious or compromised browser can spoof the referer field, but a basic XSS attack can be detected and prevented.

9.20 What is multi-factor authentication? How does it help safeguard against stolen passwords?

Answer: In multi-factor authentication (with two-factor authentication as a special case), where multiple independent *factors* (that is, pieces of information or processes) are used to identify a user. The factors should not share a common vulnerability; for example, if a system merely required two passwords, both could be vulnerable to leakage in the same manner (by network sniffing, or by a virus on the computer used by the user, for example). In addition to secret passwords, which serve as one factor, one-time passwords sent by SMS to a mobile phone, or a hardware token that generates a (time-based) sequence of one-time passwords, are examples of extra factors.

9.21 Consider the Oracle **Virtual Private Database** (**VPD**) feature described in Section 9.7.5, and an application based on our university schema.

a. What predicate (using a subquery) should be generated to allow each faculty member to see only *takes* tuples corresponding to course sections that they have taught?

b. Give an SQL query such that the query with the predicate added gives a result that is a subset of the original query result without the added predicate.

c. Give an SQL query such that the query with the predicate added gives a result containing a tuple that is not in the result of the original query without the added predicate.

Answer:

a. The following predicate can be added to queries on *takes*, to ensure that each faculty member only sees *takes* tuples corresponding to course sections that they have taught; the predicate assumes that *syscontext.user_id()* returns the instructor identifier.

```
exists (select *
        from teaches
        where teaches.ID=syscontext.user_name() and
              teaches.course_id= takes.course_id and
              teaches.section_id= takes.section_id and
              teaches.year= takes.year and
              teaches.semester= takes.semester)
```

b. The following query retrieves a subset of the answers, due to the above predicate:

```
select * from takes;
```

c. The following query gives a result tuple that is not in the result of the original query:

select count(*) **from** *takes*;

The aggregated function above can be any of the aggregate functions, such as sum, average, min or max on any attribute; in the case of min or max the result could be the same if the person executing the query is authorized to see the tuple corresponding to the min or max value. For count, sum, and average, the values are likely to be different as long as there is more than one section.

9.22 What are two advantages of encrypting data stored in the database?

Answer:

a. Unauthorized users who gain access to the OS files in which the DBMS stores the data cannot read the data.

b. If the application encrypts the data before it reaches the database, it is possible to ensure privacy for the user's data such that even privileged users like database administrators cannot access other users' data.

9.23 Suppose you wish to create an audit trail of changes to the *takes* relation.

a. Define triggers to create an audit trail, logging the information into a relation called, for example, *takes_trail*. The logged information should include the user-id (assume a function *user_id*() provides this information) and a timestamp, in addition to old and new values. You must also provide the schema of the *takes_trail* relation.

b. Can the above implementation guarantee that updates made by a malicious database administrator (or someone who manages to get the administrator's password) will be in the audit trail? Explain your answer.

Answer:

a. **Schema for the *takes_trail* table**

```
takes_trail(user_id integer, timestamp datetime, operation_code integer,
            new_ID, new_course_id, new_sec_id, new_year, new_sem, new_grade
            old_ID, old_course_id, old_sec_id, old_year, old_sem, old_grade
```

Trigger for INSERT

```
create trigger takes_insert after insert on takes
referencing new row as nrow
for each row
begin
    insert into takes_trail values (user_id(), systime(), 1,
        nrow.ID, nrow.course_id, nrow.sec_id, nrow.year, nrow.sem, nrow.grade
        null, null, null, null, null, null);
end
```

Trigger for UPDATE

```
create trigger takes_update after update on takes
referencing old row as orow, referencing new row as nrow
for each row
begin
    insert into takes_trail values (user_id(), systime(), 2,
        nrow.ID, nrow.course_id, nrow.sec_id, nrow.year, nrow.sem, nrow.grade
        orow.ID, orow.course_id, orow.sec_id, orow.year, orow.sem, orow.grade);
end
```

Trigger for DELETE

```
create trigger takes_delete after delete on takes
referencing old row as orow
for each row
begin
    insert into account_trail values (user_id(), systime(), 3,
        null, null, null, null, null, null);
        orow.ID, orow.course_id, orow.sec_id, orow.year, orow.sem, orow.grade);
end
```

b. No. Someone who has the administrator privileges can disable the trigger and thus bypass the trigger based audit trail.

9.24 Hackers may be able to fool you into believing that their Web site is actually a Web site (such as a bank or credit card Web site) that you trust. This may be done by misleading email, or even by breaking into the network infrastructure and rerouting network traffic destined for, say mybank.com, to the hacker's site. If you enter your user name and password on the hacker's site, the site can record it, and use it later to break into your account at the real site. When you use a URL such as https://mybank.com, the HTTPS protocol is used to prevent such attacks. Explain how the protocol might use digital certificates to verify authenticity of the site.

Answer: In the HTTPS protocol, a Web site first sends a digital certificate to the user's browser. The browser decrypts the digital certificate using the stored public key of the trusted certification authority and displays the site's name from the decrypted message. The user can then verify if the site name matches the one he/she intended to visit (in this example mybank.com) and accept the certificate. The browser then uses the site's public key (that is part of the digital certificate) to encrypt user's data. Note that it is possible for a malicious user to gain access to the digital certificate of mybank.com, but since the user's data (such as password) is encrypted using the public key of mybank.com, the malicious user will not be able to decrypt the data.

9.25 Explain what is a challenge–response system for authentication. Why is it more secure than a traditional password-based system?

Answer:

In a challenge-response system, a secret password is issued to the user and is also stored on the database system. When a user has to be authenticated, the database system sends a challenge string to the user. The user encrypts the challenge string using his/her secret password and returns the result. The database system can verify the authenticity of the user by decrypting the string with the same secret password and checking the result with the original challenge string.

The challenge-response system is more secure than a traditional password-based system since the password is not transferred over the network during authentication.