CS303: Assignment 3

B Siddharth Prabhu
200010003@iitdh.ac.in

17 October 2022

1 Answers to Question 1: Database Design - Automobile Company

1.1 Assumptions from given information

Before designing the database, we must make note of the assumptions to be made about this database:

- The table 'vehicle' stores info about cars owned by individuals, which are related to some dealers. These vehicles are of a certain model, made by some brand. A given model has its set of available options, and **a vehicle can only be related to options corresponding to its model id**.
- To illustrate the above point, consider models X and Y, such that Y has no roofless option, while the former can be roofless. Then, a certain vehicle of model Y should not be associated with any roofless option, since it is not available for model Y. This also follows from the question, where "individual cars can have some (or none) of the **available** options".
- Assume that IDs exist for all entities. This allows for a more systematic database.

1.2 Relational Schema

The tables in the autoDB database would be as follows:

```
brand (b_id, name)
creates (b_id, model_id)
model (model_id, name)
features (option_id, model_id)
options (option_id, description)
equipped (option_id, model_id, VIN)
has (VIN, model_id)
vehicle (VIN)
sells (VIN, d_id)
dealer (d_id, name, ph_no, street, city, PIN)
buys (VIN, c_id)
customer (c_id, name, ph_no, street, city, PIN)
```

The integrity constraints (Primary/Foreign keys) have been tabulated in a later subsection.

1.3 Entity-Relationship Diagram

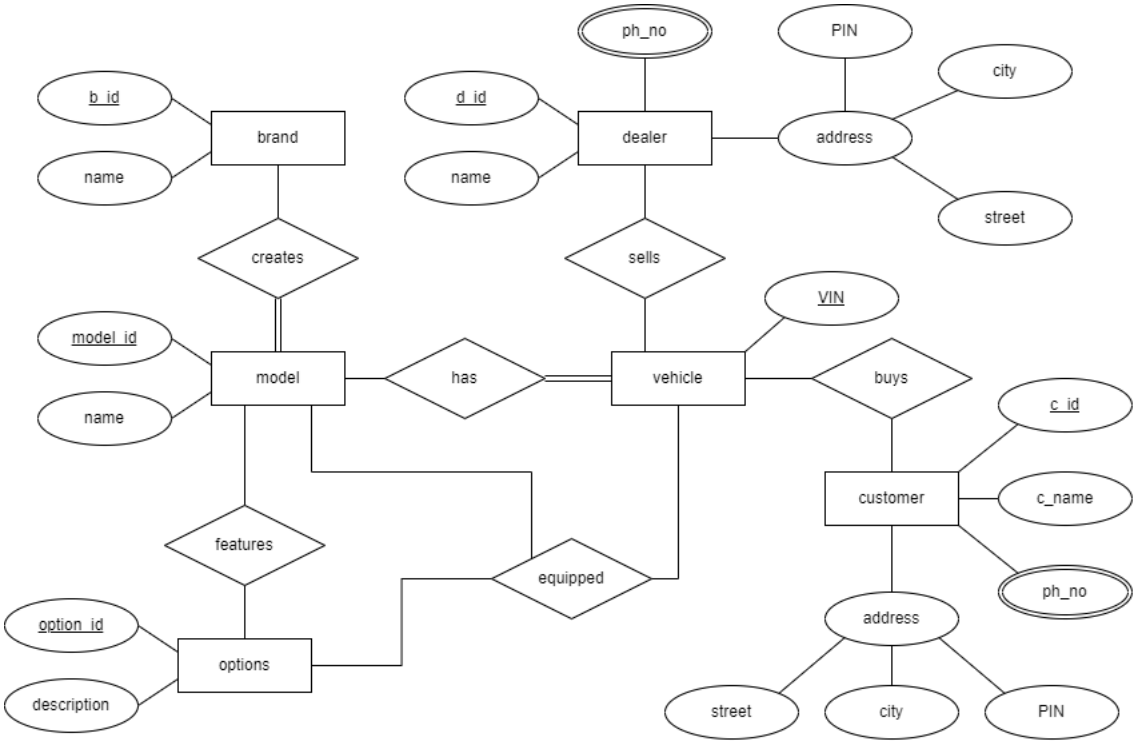

Figure 1: Entity-Relationship Diagram of the autoDB database

An alternate design could be made by connecting the equipped relation to an aggreagated entity, formed by "joining" the tables of model, features, and option. Or, we could connect equipped only to vehicle and features. In both of these procedures, the equipped relation is able to access model_id.

1.4 Integrity Constraints

Table	Primary Key (PK)	Foreign Key (Referencing Table)	Not Null
brand	b_id	-	name
creates	b_id, model_id	(b_id) references brand, (model_id) references model	-
model	model_id	-	name
features	model_id, option_id	(model_id) references model, (option_id) references options	-
options	option_id	-	description
equipped	option_id, model_id, VIN	(model_id) references model, (option_id) references options, (VIN) references vehicle	-
vehicle	VIN	-	-
sells	d_id, VIN	(VIN) references vehicle, (d_id) references dealer	-
dealer	d_id	-	name, ph_no
customer	c_id	-	c_name, ph_no
buys	c_id, VIN	(VIN) references vehicle, (c_id) references customer	-

Table 1: Integrity Constraints in autoDB schema

2 Answers to Question 2: Database Design - Delivery Company

2.1 Assumptions from given information

Before designing the database, we must make note of the assumptions to be made about this database:

- A customer can either ship or receive a package, or even do both. Each package goes through a number of locations, some of which may be modes of transport, while others are places like airports or warehouses.
- Since location ID may not be maintained by the same operating agency for all of these possible "locations", we must also keep an attribute for location type called 'loc_type', that has the corresponding type of location. Possible values for it include 'truck', 'plane', or even 'airport' or 'warehouse'. Hence, the primary key of the location table must be a combination of ID and type.
- Assume that IDs exist for all entities. This allows for a more systematic database.
- For modes of transport as locations, we will consider 'place' as place of origin, since we could say that the package has reached the destination of that part of the journey when it either gets delivered, or gets onboard the next mode of transport, or gets stored in a warehouse.
- Each entry in 'stores' keeps track of the entry and exit time of a package through some facility or mode of transport. This entry is identified using a combination of location ID, location type, and package ID.

2.2 Relational Schema

The tables in the deliveryDB database would be as follows:

```
customer (c_id, name, ph_no, street, city, country, PIN)
ships (c_id, shipping_time, p_id)
receives (c_id, delivery_time, p_id)
package (p_id, size, weight)
stores (p_id, loc_id, loc_type, entry_time, exit_time)
location (loc_id, loc_type, city, country)
```

The integrity constraints (Primary/Foreign keys) have been tabulated in a later subsection.
(contd. on next page)

2.3 Entity-Relationship Diagram

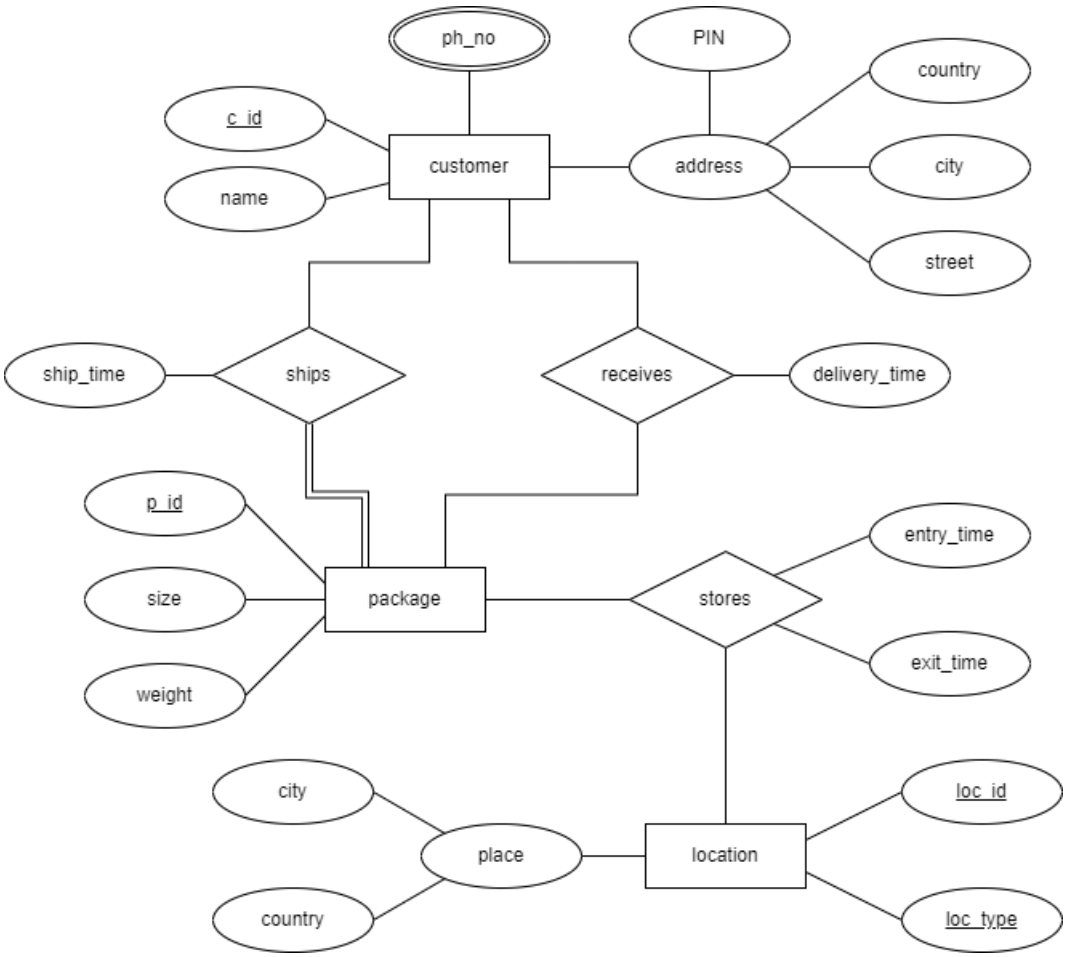

Figure 2: Entity-Relationship Diagram of the deliveryDB database

2.4 Integrity Constraints

Table	Primary Key (PK)	Foreign Key (Referencing Table)	Not Null
customer	c_id	-	name, ph_no
ships	c_id, p_id	(c_id) references customer, (p_id) references package	ship_time
receives	c_id, p_id	(c_id) references customer, (p_id) references package	delivery_time
package	p_id	-	weight, size
stores	p_id, loc_id, loc_type	(p_id) references package, (loc_id, loc_type) references location	entry_time, exit_time
location	loc_id, loc_type	-	city, country

Table 2: Integrity Constraints in deliveryDB schema

3 Answer to Question 3: Form Variable Loophole

Problem Statement

Consider a carelessly written web application for an online-shopping site, which stores the price of each item as a hidden form variable in the Web page sent to the customer; when the customer submits the form, the information from the hidden form variable is used to compute the bill for the customer. What is the loophole in this scheme?

Answer

A hacker can access the HTML source code of the page using methods like 'Inspect Element', and they **could modify the value of the item** prices by changing the hidden variable value. Then, since the bill is computed using the hidden form variable, the bill would use these modified values. Hence, storing prices on the client side as hidden form variables is highly insecure and irresponsible.

This may raise the question about why hidden form variables are used in the first place. However, we must note that their purpose is not security, but data abstraction. Thus, this method of storing information is inadequate and inapplicable for purposes such as the one mentioned above.

4 Answer to Question 4: Session Risks

Problem Statement

Consider another carelessly written Web application, which uses a servlet that checks if there was an active session, but does not check if the user is authorized to access that page, instead depending on the fact that a link to the page is shown only to authorized users. What is the risk with this scheme?

Answer

Unauthorized users can still access the page via the URL in the browser. If there is no session authorization check in place at the required page, then the unauthorized can read all the data from it. Such users may become aware of the URLs by different means that can't be prevented.

So, we must make sure that even knowledge of the URLs does not give access to users without authorization. (Although we may be able to use HTTP referer attribute to block access from invalid pages of a website, even this is set at the browser. Hence, this could be exploited as well). Hence, separate checks for authorization must be in place for all pages that are user-specific. Simply checking for if a session is active is not sufficient.

5 Answer to Question 5: Digital Certificates for Authentication

Problem Statement

Hackers may be able to fool you into believing that their website is actually a credible website (such as a bank or credit card website). This may be done by misleading emails, 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 username 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

A digital certificate contains information such as: user's name, company/department of user, IP address/serial number of device, copy of the public key from a certificate holder, validity duration of the certificate, domain of certificate authorization. A general idea of its working is as follows:

Digital Certificates are used in public key cryptography, which depends on key pairs: 1 private key to be held by the owner (used for signing and decrypting), and 1 public key, which is used for encrypting data sent to the public key owner, and for authenticating the certificate holder's signed data. The digital certificate **enables entities to share their public key** so it can be authenticated.

If a website has a valid certificate, it means that a certificate authority has taken steps to verify that the web address really belongs to that organization. If one types a URL or follows a link to a secure website, the browser will check the certificate (using HTTPS - HyperText Transfer Protocol) for characteristics such as:

- The website address matches the address on the certificate.
- The certificate is signed by a certificate authority that the browser recognizes as a "trusted" authority.

When a user wants to send confidential information to a Web server, such as a credit-card number for an online transaction, the browser will access the public key in the server's digital certificate to verify its identity.

Uses of Digital Certificates apart from website authenticity checks:

- in secure emails for users to be able to identify each other. **(solves problem described in the question)**
- for electronic document signing; the sender digitally signs the email, and the recipient verifies the signature.
- for key exchange (used in public key encryption and authentication of digital signatures).

6 Answer to Question 6: Authenticator Servlet

Problem Statement

Write a servlet and associated HTML code for the following simple application: A user logins using userID and password and then a servlet authenticates the user (based on user IDs and passwords stored in a database relation), and sets a session variable called userid after authentication.

Answer

For this, we would need the code files `Input.jsp` , `Result.jsp` , `LoginServlet.jsp` , as follows:

Firstly, `Input.jsp` :

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                                "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Log In Form</title>
</head>
<body>
<h1> Welcome to MPQMD Authenticator </h1>
<form action="LoginServlet" method="post">
    <table style="with: 50%">
        <tr>
            <td>User ID</td>
            <td><input type="text" name="input_id" required></td>
        </tr>
        <tr>
            <td>Password</td>
            <td><input id="input_pwd" name="input_pwd"
                type="password" class="input" required></td>
        </tr>
    </table>
    <input type="submit" value="Submit"></form>
</body>
</html>
```

Next, `Result.jsp` :

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Logged in</title>
</head>
<body>
 <%
 response.setContentType("text/html");
 if(session!=null){
 String name = (String)session.getAttribute("userid");
 out.print("Hello, "+name+" ! You Are Logged In.");
 }
 else
 {
     out.print("Please login first");
     request.getRequestDispatcher("login.html").include(request, response);
 }
 out.close();
 %>
</body>
</html>
```

Finally, `LoginServlet.java` :

```
import java.io.*;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;
    public LoginServlet() {
        super();
    }
        protected void doPost(HttpServletRequest request,
                              HttpServletResponse response)
                throws ServletException, IOException {
                try
                {
                //getting input values from jsp page
                String id = request.getParameter("input_id");
                String pwd = request.getParameter("input_pwd");
```

```
Connection con = null;
String url = "jdbc:mysql://localhost:3306/logindb";
String username = "auth123"; //MySQL username
String password = "mepregunto"; //MySQL password

Class.forName("com.mysql.jdbc.Driver");
con = DriverManager.getConnection(url, username, password);
System.out.println("Printing connection object "+con);
HttpSession session = request.getSession();
PrintWriter out = response.getWriter();

//Prepared Statement to get hashed password
PreparedStatement st = con.prepareStatement(
    "SELECT pass FROM userlist WHERE userID = ?;"
    );
st.setString(1, id);
ResultSet rset = st.executeQuery();

if(rset.next()){
String pass_db = rset.getString("pass");
System.out.println("pass hash = "+ pass_db);
PreparedStatement st_in = con.prepareStatement(
        "SELECT SHA2(? , 256) AS hash;"
        );
    st_in.setString(1, pwd);
    ResultSet rset_in = st_in.executeQuery();
rset_in.next();
String pass_in = rset_in.getString("hash");
//                System.out.println("input hash = "+ pass_in);

if(pass_in.equals(pass_db))
{
    RequestDispatcher rd = request.getRequestDispatcher("Result.jsp");
    session.setAttribute("userid", id);
    rd.forward(request, response);
} else {
    RequestDispatcher rd = request.getRequestDispatcher("Input.jsp");
    out.println("<font color=red> Password is wrong. </font>");
    rd.include(request, response);
}

} else {
System.out.println("no such user!!!");
RequestDispatcher rd = request.getRequestDispatcher("Input.jsp");
out.println("<font color=red> User ID does not exist </font>");
rd.include(request, response);
}
```

```
        out.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

These code segments have been tested for different exception cases and normal cases. Screenshots and other code details aren't included here for the sake of brevity. (Note that we are encrypting the passwords using SHA256 hashing! It is completely inappropriate to store passwords as raw text in a database.)

Find below the SQL code used for the initial setup:

```sql
-- creation of new user, and granting of privileges
CREATE USER 'auth123'@'localhost' IDENTIFIED BY 'mepregunto';
CREATE DATABASE logindb;
GRANT ALL PRIVILEGES ON logindb.* TO 'auth123'@'localhost';

-- After logging in to new user, create table
SHOW DATABASES;
USE logindb;
CREATE TABLE userlist(
    userID VARCHAR(30) PRIMARY KEY,
    pass VARCHAR(64) NOT NULL
);
DESCRIBE userlist;

-- load users into the database
INSERT INTO userlist VALUES('10004', SHA2('password', 256));
INSERT INTO userlist VALUES('10003', SHA2('wenomechainsama', 256));
INSERT INTO userlist VALUES('10012', SHA2('tumajarbisaun', 256));
INSERT INTO userlist VALUES('10053', SHA2('wifelinlooof', 256));
INSERT INTO userlist VALUES('20012', SHA2('eselifterbraun', 256));
INSERT INTO userlist VALUES('10013', SHA2('alanturing', 256));
INSERT INTO userlist VALUES('10001', SHA2('mepregunto', 256));
INSERT INTO userlist VALUES('10002', SHA2('quemedirias', 256));
INSERT INTO userlist VALUES('10005', SHA2('sipudieras', 256));
INSERT INTO userlist VALUES('10006', SHA2('hablarwenom', 256));
SELECT * FROM userlist;
```

7 Answer to Question 7: XSS Attacks

Problem Statement

What is an XSS attack? Explain how it works. What measures must be taken to detect or prevent XSS attacks?

Answer

Cross-Site Scripting (XSS) Attacks are attacks constituting injection of malicious scripts into trusted websites, generally as a browser-side script, to a different end-user. They can occur anywhere a web application uses input from a user within the output it generates, without validating or encoding it. While it usually persists in JavaScript, XSS can take place in any client-side language.

To carry out a cross site scripting attack, an attacker injects a malicious script into user-provided input. Attackers can also carry out an attack by modifying a request. Sometimes, users may be lured to malicious websites. If the web app is vulnerable to XSS attacks, the user-supplied input will execute as code. For example, in the request below, the script displays a message box with the text "xss.":

```
http://www.site.com/page.php?var=<script>alert('xss');</script>
```

An XSS Attack can be triggered in different ways: When the page loads, or When a user hovers over specific elements of the page. Scripts could be of the form:

```
<script>doSomethingEvil();</script>
```

Measures to prevent or detect XSS attacks:

- Don't trust any user input directly: In general, it is a good idea to sanitize inputs that are obtained from the user. This can be done by using encoding libraries to encode characters that could affect execution context.
- Use escaping/encoding techniques.
- Set the HTTPOnly flag for cookies. This ensures that cookies cannot be accessed by client side scripts, effectively blocking XSS attacks.
- Context security policies can be used to specify access permissions to client side resources.
- HTTP Referer should be used to check the source page from where the user is accessing the current page.
- Session should be restricted to the IP Address where it was authenticated, so that login isn't possible from other systems even in the case of cookie leaks.

What these points essentially mean is that: Anytime some input is taken from the user, HTML tag insertion should be averted. Detection and stripping of such tags must be done. Even an image may be dangerous if image display software can be exploited! Lastly, never use "GET" method to perform updates.