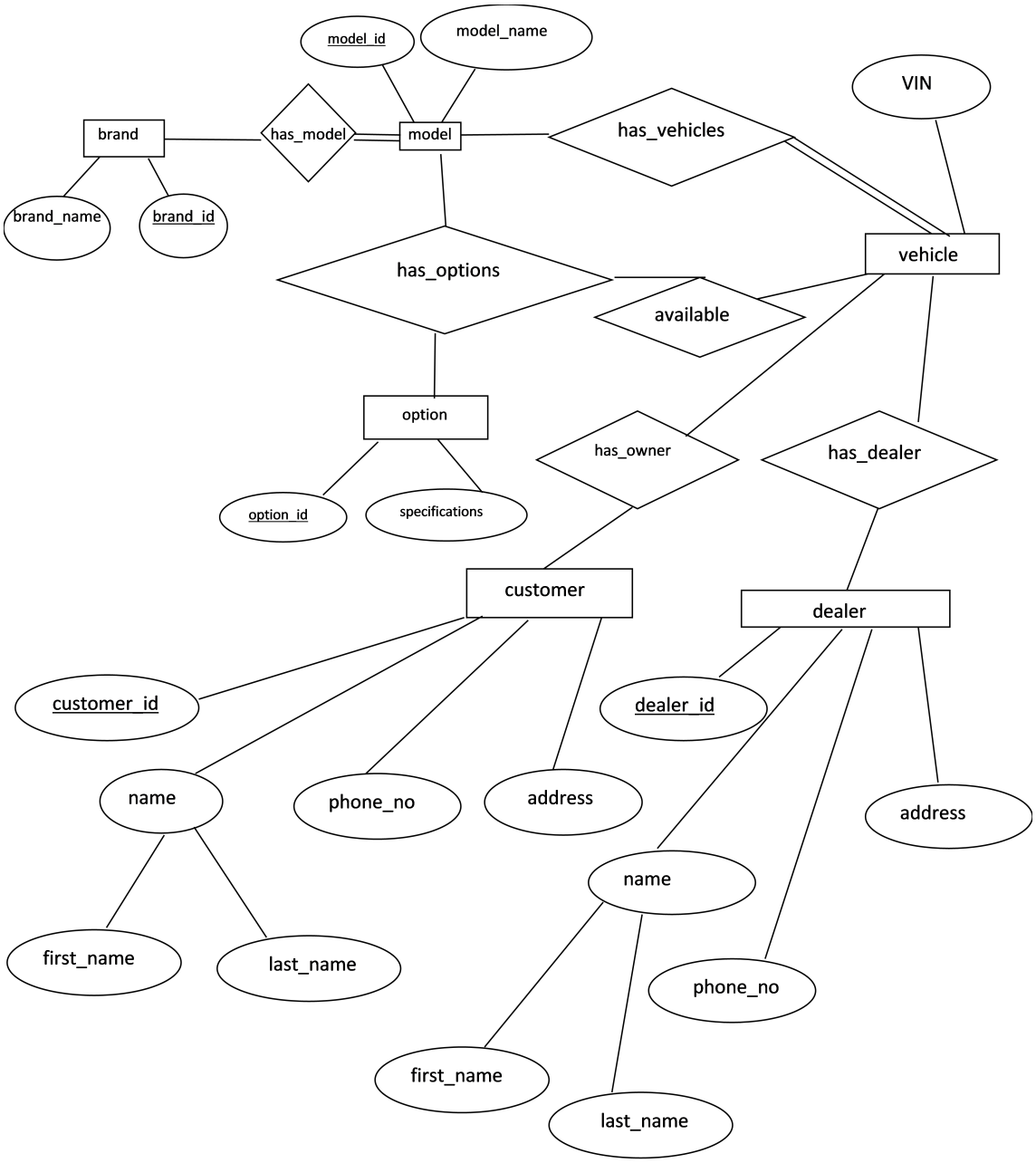
CS-303 : Assignment-3

Monu Kumar Soyal-200010029

Answer-1

ER Model :-

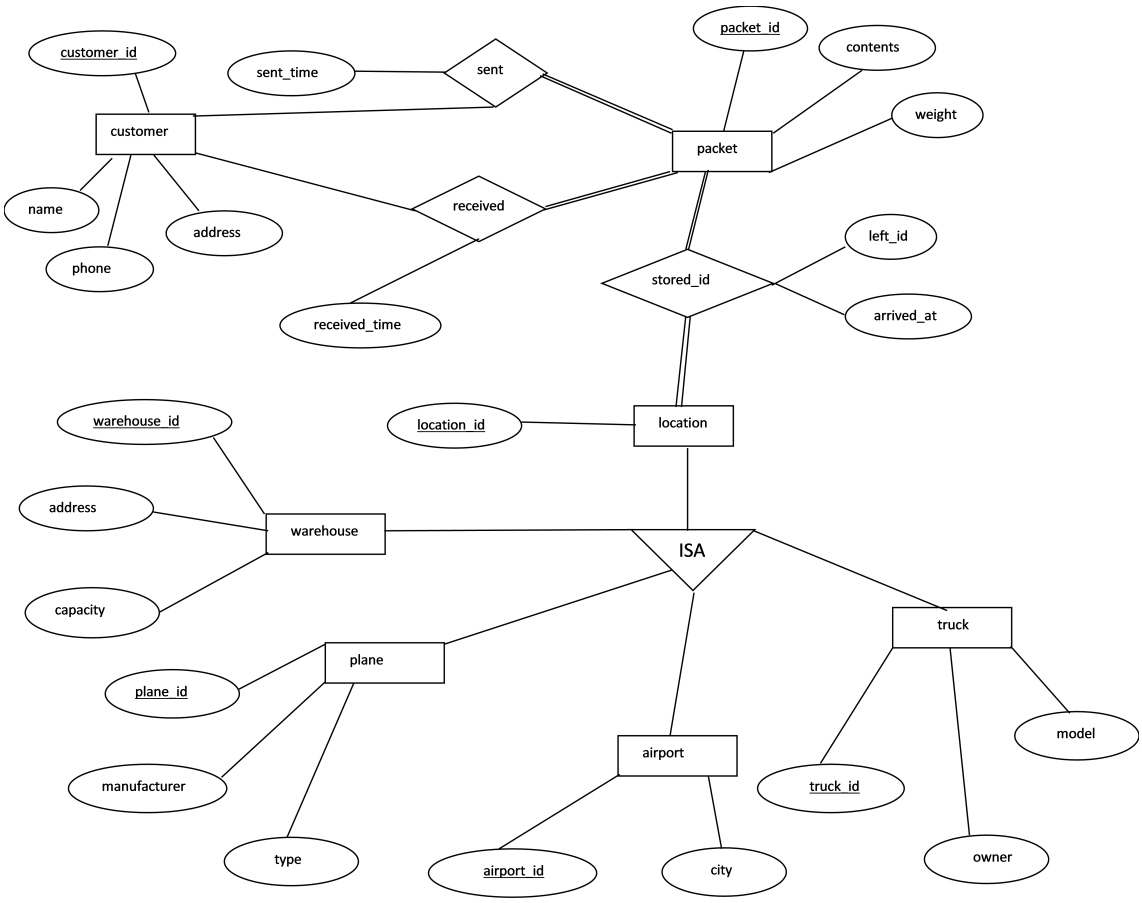


Schema :-

brand (<u>brand_id</u>, brand_name)
model (<u>model_id</u>, model_name)
vehicle (<u>VIN</u>)

option (option_id, specifications)
customer (customer_id, name, address, phone_no)
dealer (dealer_id, name, address, phone_no)
has_model (brand_name, model_id,
foreign key brand_name **references** brand,
foreign key model_id **references** model)
has_vehicle (model_id, VIN,
foreign key VIN **references** vehicle,
foreign key model_id **references** model)
has_option (model_id, option_id,
foreign key option_id **references** option,
foreign key model_id **references** model)
available (VIN, model_id, option_id,
foreign key VIN **references** vehicle,
foreign key (model_id, option_id) **references** has_option)
has_dealer (VIN, dealer_id,
foreign key dealer_id **references** dealer,
foreign key VIN **references** vehicle)
customer (VIN, customer_id,
foreign key customer_id **references** customer,
foreign key VIN **references** vehicle)

Answer 2



Schemas :-

customer (customer_id, name, address, phone)
packet (packet_id, weight, contents)
location (location_id)
truck (truck_id, owner, model)

plane (plane_id, type, manufacturer)
airport (airport_id, city)
warehouse (warehouse_id, address, capacity)
stored_id (packet_id, location_id, arrived_at, left_id,
    foreign key packet_id **references** packet,
    foreign key location_id **references** location)
received (customer_id, packet_id, received_time,
    foreign key packet_id **references** packet,
    foreign key customer_id **references** customer)
sent (customer_id, packet_id, sent_time,
    foreign key packet_id **references** packet,
    foreign key customer_id **references** customer)

Answer 3

A hacker can modify the web page's HTML source code, replace the value of the secret variable price with any value they choose, and then use the altered web page to submit an order. The pricing of the product would then be determined by the Web application using the user-modified value.

Answer 4

Even though only authorised users can see the connection to the page, an unauthorised user might somehow learn about the link's existence (for example, from an unauthorised user, or via Web proxy logs). The user can then log in to the system and visit the illegal page by entering its URL in the browser.

The user will be able to view the page's outcome if the authorisation check for the user was unintentionally left out of it. By guaranteeing that the referer value is from a legitimate page of the website, the HTTP referer attribute can be utilised to prevent a careless attempt to exploit such weaknesses. However, the referer attribute is set by the browser and can be faked.

Answer 5

A website sending via HTTPS first transmits a cryptographic certificate to the user's browser. Using the trusted certification authority's public key that is stored in the browser, the digital certificate is decrypted, and the name of the website is then displayed.

The user can then confirm that the website is the one they wanted to visit (in this case, mybank.com), if so, they can accept the certificate. After that, the browser encrypts the user's data using the website's public key, which is contained in the digital certificate. Although it is possible for a bad user to obtain access to mybank.com's digital certificate, the malicious user won't be able to read the user's data because it is encrypted with mybank.com's public key.

Answer 6

Login.jsp

```
1
2  <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
3      pageEncoding="ISO-8859-1"%>
4  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
5  <html>
6  <head>
7  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8  <title>User Form</title>
9  </head>
10 <body>
11         <h1>User Details </h1>
12         <form action="UserServlet" method="post">
```

```
                <table style="width: 50%">
                        <tr>
                                <td>User Id</td>
                                <td><input type="text" name="id" /></td>
                        </tr>
                        <tr>
                                <td>User Password</td>
                                <td><input type="password" name="password" /></td>
                        </tr>
                </table>
                <input type="submit" value="Submit" />
        </form>
</body>
</html>
```

UserSurvlet.java

```
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;


@WebServlet ("/UserServlet")
public class UserServlet extends HttpServlet {
private static final long serialVersionUID = 1L ;
public UserServlet() {
    super();
}
protected void doPost(HttpServletRequest request, HttpServletResponse response )
throws ServletException , IOException {
    String id = request.getParameter("id") ;
    String password = request.getParameter("password") ;

    Connection con = null ;
    String url = "jdbc:mysql://localhost:3306/LoginDatabase";

    String username = "logindatabaseDB0029";
    String jdbcpassword = "1234567890";
    Class.forName ("com.mysql.jdbc.Driver") ;
    con = DriverManager.getConnection(url,username, jdbcpassword) ;

    PreparedStatement st = con.prepareStatement("Select* from users WHERE username =? and password =?)")
    st.setString(1, id) ;
    st.setString(2, password) ;
    ResultSet result = st.executeQuery() ;

    if(result.next()) {
        HttpSession httpSession = request.getSession();
        httpSession.setAttribute("userId", id) ;
        request.getRequestDispatcher("HomePage.jsp").forward(request, response);
    }
```

```
43      else {
44          request.getRequestDispatcher("Login_Error.jsp").forward (request, response ) ;
45      }
46  }
47  }
48
```

Answer 7

Definition:

A online security flaw called cross-site scripting, or XSS, enables an attacker to tamper with how users interact with a susceptible application. It enables an attacker to get around the same origin policy, which is intended to keep various websites separate from one another. Cross-site scripting flaws typically give an attacker the ability to pretend to be a victim user, execute any operations they are capable of performing, and access any of the user's data. The attacker may be able to fully manage all of the functionality and data of the application if the target user has privileged access within it.

How does XSS work

Attackers can insert their own script, which is subsequently performed by the victim's browser, when they inject their own code into a web page, which is often accomplished by taking advantage of a software flaw on the website.

Due to the fact that JavaScript executes on the victim's browser page, important information about the authorised user can be taken from the session, allowing a bad actor to basically target site administrators and compromise an entire website.

When the vulnerability is present on the majority of a website's publicly accessible pages, cross-site scripting attacks are frequently used in this situation. In this scenario, hackers can inject their code to target website users by including their own advertisements, phishing invitations, or other malicious information.

Measures must be taken to detect or prevent XSS attacks

A web application can safeguard itself from Cross-Site Scripting problems in a variety of ways. Some of them consist of

- **Blacklist filtering -** A screening method that only partially shields the website from XSS problems is simple to create. Based on a known list of finite XSS vectors, it operates. The majority of XSS vectors, for instance, make use of event listener properties like onerror, onmouseover, onkeypress, etc. This fact makes it possible to parse the HTML attributes provided by users and these event listeners attributes.
- **Whitelist filtering -** Filtering based on whitelists is the reverse of filtering based on blacklists. Whitelist filtering lists out a set of predetermined HTML tags and attributes rather than listing out hazardous attributes and sanitising user HTML with this list. All other entities will be screened out, and those that are known to be safe will be kept.

  This drastically minimises the likelihood of XSS and only allows XSS to occur when the filter contains a flaw that allows some dangerous entities to be treated as safe. Both the client-side and the server-side can perform this filtering. The most widely used filter in contemporary web applications is whitelist filtering.
- **Contextual Encoding -** The other widely used mitigating strategy is to treat all user-provided information as textual information rather than HTML content, even when it is. This can be accomplished by encoding user data with HTML entities.
- **Input Validation -** For each request parameter, or piece of user-generated content, a regular expression is applied in the input validation process. It is only permitted if the content successfully navigates a safe regular expression. If not, the server-side request will be rejected with a 400 response code.
- **Content Security Policy -** The use of CSP, or Content Security Policy Headers, is supported by current browsers. One can define a list of domains from which JavaScript material can only be loaded using these headers. CSP headers will deny the request if the user tries to add a weak JavaScript.