In this example, we connect to a MySQL database using the JDBC driver **com.mysql.cj.jdbc.Driver**. We establish a connection by providing the database URL, username, and password. Then, we create a **Statement** object and execute a SQL query to select all records from the "customers" table. The result set is obtained by calling **executeQuery()** on the **Statement** object. We iterate over the result set using **resultSet.next()** and retrieve values using the column names or indexes.

Finally, we close the resources in the **finally** block to release the database connection and other JDBC objects.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcExample {
    public static void main(String[] args) {
        // JDBC connection parameters
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "myuser";
        String password = "mypassword";

        // Connection, Statement, and ResultSet variables
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        try {
            // Step 1: Load and register the JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Step 2: Establish the connection
            connection = DriverManager.getConnection(url, username, password);

            // Step 3: Create a statement
            statement = connection.createStatement();

            // Step 4: Execute a query
            String sqlQuery = "SELECT * FROM customers";
```

```
            resultSet = statement.executeQuery(sqlQuery);

            // Step 5: Process the result set
            while (resultSet.next()) {
                int id = resultSet.getInt("id");
                String name = resultSet.getString("name");
                String email = resultSet.getString("email");

                System.out.println("ID: " + id + ", Name: " + name + ", Email: " +
email);
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            // Step 6: Close the resources
            try {
                if (resultSet != null) {
                    resultSet.close();
                }
                if (statement != null) {
                    statement.close();
                }
                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

In this example, we first create a table named "customers" with three columns: "id" of type INT (integer), "name" of type VARCHAR (variable-length string), and "email" of type VARCHAR.
Then, we insert three records into the "customers" table using the **INSERT INTO** statement.
Finally, we select all records from the "customers" table using the **SELECT** statement, which retrieves all columns and rows from the table.

```
-- Create a table
CREATE TABLE customers (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  email VARCHAR(100)
);

-- Insert data into the table
INSERT INTO customers (id, name, email) VALUES
  (1, 'John Doe', 'john@example.com'),
  (2, 'Jane Smith', 'jane@example.com'),
  (3, 'Mike Johnson', 'mike@example.com');

-- Select all records from the table
SELECT * FROM customers;
```

```
<!DOCTYPE html>
<html>
<head>
    <title>Customer Records</title>
</head>
<body>
    <h1>Customer Records</h1>

    <table>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Email</th>
        </tr>
        <!-- Table rows will be dynamically populated -->
    </table>

    <script>
        // Use AJAX to fetch customer records from the server
        fetch('customers')
            .then(response => response.json())
            .then(data => {
                // Loop through the records and populate the table
                data.forEach(customer => {
                    const row = document.createElement('tr');
                    row.innerHTML = `
                        <td>${customer.id}</td>
```

```
                <td>${customer.name}</td>
                <td>${customer.email}</td>
            `;
            document.querySelector('table').appendChild(row);
        });
    })
    .catch(error => {
        console.error('Error:', error);
    });
</script>
</body>
</html>
```

In this example, the **index.html** file contains an HTML table where the customer records will be dynamically populated using JavaScript. The script uses AJAX to fetch the customer records from the server.
The **web.xml** file is a configuration file for the Java web application. It includes mappings for the default servlet and the servlet responsible for handling customer requests. The **CustomerServlet** is defined with the servlet class **com.example.CustomerServlet** and is mapped to the URL pattern **/customers**.
You need to create a servlet class (e.g., **CustomerServlet**) that will handle the **/customers** request and retrieve the customer records from the database. You can use JDBC within the servlet to execute the SQL query and retrieve the records.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                    http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
    id="WebApp_ID" version="4.0">

  <display-name>CustomerWebApp</display-name>

  <!-- Mapping for the default servlet -->
  <servlet>
    <servlet-name>DefaultServlet</servlet-name>
    <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>0</param-value>
    </init-param>
```

```
        <init-param>
            <param-name>listings</param-name>
            <param-value>false</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>DefaultServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <!-- Servlet mapping for handling customer requests -->
    <servlet>
        <servlet-name>CustomerServlet</servlet-name>
        <servlet-class>com.example.CustomerServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CustomerServlet</servlet-name>
        <url-pattern>/customers</url-pattern>
    </servlet-mapping>
</web-app>
```