What is a database?

A database is an organized collection of structured data stored and accessed electronically from a computer system. It serves as a central repository for storing and managing data, making it easy to retrieve, update, and analyze information as needed.

What is a Database Management System (DBMS)?

A Database Management System (DBMS) is software that provides an interface for users to interact with databases. It manages the storage, retrieval, and manipulation of data in databases, ensuring data integrity, security, and performance.

What is SQL?

SQL, or Structured Query Language, is a programming language used for managing and manipulating relational databases. It provides a standardized way to interact with databases, enabling users to perform various operations such as querying, updating, inserting, and deleting data.

SQL is essential for tasks such as retrieving information from databases, updating existing data, inserting new data records, and deleting unnecessary data. It's widely used across different industries for data management and analysis.

Why do we use SQL?

SQL is used because of its versatility and power in handling structured data. Here are some key reasons why SQL is widely used:

1. **Data Retrieval:** SQL allows users to retrieve specific information from databases based on defined criteria, making it easy to access relevant data.
2. **Data Manipulation:** SQL enables users to manipulate data within databases, including adding, modifying, and deleting records.
3. **Data Definition:** SQL provides commands for defining the structure and schema of databases, including creating tables, indexes, and constraints.
4. **Data Control:** SQL includes commands for managing access to databases, such as granting and revoking permissions.

Overall, SQL simplifies the process of interacting with databases, making it easier to manage and analyze large volumes of data efficiently.

When was it introduced?

SQL was introduced in the early 1970s by IBM researchers Raymond Boyce and Donald Chamberlin. Initially developed as a research project at IBM, SQL quickly gained popularity and became the standard language for relational databases.

Types of SQL:

Data Definition Language (DDL):

DDL is used to define the structure and schema of a database. It includes commands for creating, altering, and dropping database objects such as tables, views, indexes, and constraints.

Example:

CREATE TABLE Students (StudentID INT PRIMARY KEY, FirstName VARCHAR(50), LastName VARCHAR(50), Age INT);

Data Manipulation Language (DML):

DML is used to manipulate data within the database. It includes commands for inserting, updating, and deleting data records.

Example:

INSERT INTO Students (StudentID, FirstName, LastName, Age) VALUES (1, 'John', 'Doe', 20);

UPDATE Students SET Age = 21 WHERE StudentID = 1;

DELETE FROM Students WHERE StudentID = 1;

Data Query Language (DQL):

DQL is used to retrieve data from the database. It includes commands for querying and retrieving data from one or more tables based on specified criteria.

Example:

SELECT * FROM Students;

Different types of applications:

SQL is used in a wide range of applications across various industries, including:

- **Web Development:** SQL is used to interact with databases in web applications, enabling dynamic content generation and user authentication.
- **Mobile App Development:** SQL is used in mobile apps for storing and retrieving data, enabling offline functionality and data synchronization.
- **Data Analysis:** SQL is used in data analysis tools and business intelligence platforms for querying and aggregating data from large datasets.
- **Enterprise Resource Planning (ERP) Systems:** SQL is used in ERP systems for managing business processes and integrating data across different departments.
- **Customer Relationship Management (CRM) Systems:** SQL is used in CRM systems for storing and managing customer data, enabling personalized marketing and customer service.

What is MySQL?

MySQL is an open-source relational database management system (RDBMS) that uses SQL for managing and manipulating data. It's widely used in web development and other applications due to its scalability, reliability, and performance.

What is a schema?

A schema in a database is a collection of database objects, including tables, views, indexes, and constraints. It defines the structure and organization of the database, including the relationships between different data elements.

What is a table?

A table in a database is a structured collection of data organized in rows and columns. Each row represents a record, and each column represents a field or attribute of the record. Tables are used to store and organize data in a relational database.

What is an entity?

An entity is a real-world object or concept that is represented in the database. It could be a person, place, thing, or event that is relevant to the application domain. Entities are typically represented as tables in a database, with each row representing a specific instance of the entity.

What is an attribute?

An attribute is a characteristic or property of an entity. It describes a specific aspect or feature of the entity and is represented as a column in a database table. Attributes define the type of data that can be stored in a particular field, such as text, numbers, dates, or binary data.

1. **Download Database Driver**: Visit the MySQL website or search online for the MySQL JDBC driver. Look for the appropriate version compatible with your MySQL server version and download the JDBC driver JAR file. For MySQL, you can typically find the driver at https://dev.mysql.com/downloads/connector/j/.
2. **Include Driver in Classpath**: After downloading the JDBC driver JAR file, include it in your Java project's classpath. This allows your Java application to access the classes and interfaces provided by the JDBC driver. You can add the JAR file to your project's build path in your IDE (like Eclipse or IntelliJ IDEA) or include it in the classpath when compiling and running your Java program from the command line.
3. **Import JDBC Packages**: In your Java code, import the JDBC packages required for database connectivity and operations. The most common packages you'll need are **java.sql.***, which contains the core JDBC classes and interfaces.
4. **Load the Driver**: Before establishing a connection to the MySQL database, you need to load the MySQL JDBC driver class into memory using **Class.forName()**. This step dynamically loads the driver class, making it available for use in your application. For MySQL, the driver class is typically **com.mysql.cj.jdbc.Driver**.
5. **Establish Connection**: Use the **DriverManager.getConnection()** method to establish a connection to the MySQL database. You'll need to provide the database URL, username, and password as parameters to this method. The database URL format for MySQL typically looks like **"jdbc:mysql://hostname:port/databasename"**, where you replace **hostname**, **port**, and **databasename** with your MySQL server details.
6. **Create Statement**: Once the connection is established, you can create a **Statement** object using the **createStatement()** method of the **Connection** object. This statement object is used to execute SQL queries against the database.
7. **Execute Query**: To execute a SQL query, use the **executeQuery()** method of the **Statement** object. Pass the SQL query string as a parameter to this method. For example, **"SELECT * FROM tablename"** to select all records from a table named **tablename**.
8. **Process Results**: If the query returns a result set (e.g., in the case of a SELECT query), you can iterate through the result set using a **ResultSet** object. Use methods like **next()** to move to the next row and **getString()**,

getInt(), or other appropriate methods to retrieve column values from the result set.

9. **Close Resources**: After you're done with the database operations, make sure to close the resources properly to release database connections and resources. Close the **ResultSet**, **Statement**, and **Connection** objects in reverse order of their creation using their **close()** methods.

```
import java.sql.*;

public class JDBCExample {
    public static void main(String[] args) {
        try {
            // Load the JDBC driver
            Class.forName("com.mysql.jdbc.Driver");

            // Establish connection
            Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydatabase", "username", "password");

            // Create statement
            Statement statement = connection.createStatement();

            // Execute query
            ResultSet resultSet = statement.executeQuery("SELECT * FROM mytable");

            // Process results
            while (resultSet.next()) {
                // Retrieve and print column values
                System.out.println(resultSet.getString("column1"));
                System.out.println(resultSet.getString("column2"));
                // Repeat for other columns as needed
```

```
            }

            // Close resources
            resultSet.close();
            statement.close();
            connection.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

JDBC with servlets

A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed through a request-response programming model. They are typically used to handle requests and generate dynamic content for web applications. In simpler terms, servlets are Java programs that run on a web server and handle incoming requests from clients, like web browsers, by generating responses dynamically. Running a servlet in Eclipse typically involves deploying your web application to a servlet container like Apache Tomcat directly from within the Eclipse IDE. Here's a step-by-step guide on how to run a servlet in Eclipse:

1. **Set Up Eclipse**:
    - Make sure you have Eclipse IDE for Java EE Developers installed. It includes support for developing web applications with servlets.
    - Ensure that you have Apache Tomcat (or any other servlet container) installed and configured in Eclipse.
2. **Create a Dynamic Web Project**:
    - Open Eclipse and create a new Dynamic Web Project by selecting "File" > "New" > "Dynamic Web Project".
    - Enter a project name and click "Next".
    - Choose the target runtime (Apache Tomcat) and click "Finish".
3. **Create a Servlet**:
    - Right-click on the **src** folder in your project and select "New" > "Servlet".
    - Enter a package name and servlet class name.
    - Click "Next", then "Finish" to create the servlet class.
4. **Write Servlet Code**:
    - Implement the **doGet()** or **doPost()** method in your servlet to handle HTTP requests.
    - You can use JDBC code within your servlet to interact with the database.
5. **Configure Servlet Mappings**:

- If you're using Servlet 2.5 or earlier, you'll need to configure servlet mappings in the **web.xml** deployment descriptor file.
- If you're using Servlet 3.0 or later, you can use annotations like **@WebServlet** to define mappings directly in the servlet class.

6. **Deploy the Project**:
    - Right-click on your project and select "Run As" > "Run on Server".
    - Choose your configured server (e.g., Apache Tomcat) and click "Finish".
    - Eclipse will deploy your web application to the servlet container and start the server.
7. **Access the Servlet**:
    - Once the server is started, you can access your servlet by navigating to its URL in a web browser.
    - The URL will typically be **http://localhost:8080/your_web_application/your_servlet_mapping**.
8. **Test the Servlet**:
    - Enter data into your HTML form (if applicable) and submit it to test the servlet's functionality.
    - Verify that the servlet interacts with the database as intended.

By following these steps, you can run a servlet in Eclipse and test its functionality within a servlet container.

```java
import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.*;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/SubmitFormServlet")

public class SubmitFormServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

String depName = request.getParameter("dep_name");

String faculty = request.getParameter("facaulty");

String classroomNo = request.getParameter("classroom_no");

// Initialize database connection

try {

Class.forName("com.mysql.cj.jdbc.Driver");

String URL = "jdbc:mysql://localhost:3306/msk";

String USERNAME = "root";

String PASSWORD = "root";
```

```
try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD)) {

// Prepare SQL statement

String sql = "INSERT INTO departments (dep_name, facaulty, classroom_no) VALUES
(?, ?, ?)";

try (PreparedStatement statement = conn.prepareStatement(sql)) {

// Set parameters

statement.setString(1, depName);

statement.setString(2, faculty);

statement.setString(3, classroomNo);

// Execute query

// int rowsAffected = statement.executeUpdate();

statement.executeUpdate();

// Redirect to doGet to display department details

response.sendRedirect(request.getContextPath() + "/SubmitFormServlet");

}

}

} catch (ClassNotFoundException | SQLException e) {

// Handle database errors

e.printStackTrace();

response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);

}

}

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

List<String> departmentDetails = new ArrayList<>();

try {

Class.forName("com.mysql.cj.jdbc.Driver");

String URL = "jdbc:mysql://localhost:3306/msk";
```

```
String USERNAME = "root";

String PASSWORD = "root";

try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD)) {

String sql = "SELECT dep_name, facaulty, classroom_no FROM departments";

try (PreparedStatement statement = conn.prepareStatement(sql)) {

ResultSet resultSet = statement.executeQuery();

while (resultSet.next()) {

String depName = resultSet.getString("dep_name");

String faculty = resultSet.getString("facaulty");

String classroomNo = resultSet.getString("classroom_no");

departmentDetails.add("Department Name: " + depName + ", Faculty: " + faculty

+ ", Classroom No: " + classroomNo);

}

}

}

} catch (ClassNotFoundException | SQLException e) {

e.printStackTrace();

response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);

}

PrintWriter out = response.getWriter();

response.setContentType("text/html");

out.println("<html><body>");

out.println("<h2>Department Details:</h2>");

for (String detail : departmentDetails) {

out.println("<p>" + detail + "</p>");

}

out.println("</body></html>");
```

```
}
}
```

JDBC WITH JSP

JSP stands for JavaServer Pages. It's a technology used to create dynamic web pages by embedding Java code into HTML pages. JSP pages are similar to regular HTML pages but can include Java code snippets, which are executed on the server before the page is sent to the client's web browser. This allows for the creation of dynamic content, such as database queries, conditional statements, and loops, within the HTML markup. JSP simplifies the process of creating dynamic web content by allowing developers to combine Java code with HTML, making it easier to generate dynamic web pages.

Here's a simple example of a JSP page that connects to a database using JDBC, takes input from an HTML form, inserts the data into a table, and then displays the entered data:

```
<%@ page import="java.sql.*" %>
<%@ page import="javax.servlet.*" %>
<%@ page import="javax.servlet.http.*" %>
<html>
<head>
<title>Insert Data</title>
</head>
<body>
<h2>Insert Data into Database</h2>
<%
    String firstName = request.getParameter("firstname");
    String lastName = request.getParameter("lastname");
```

```
    // JDBC driver name and database URL
    String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    String DB_URL = "jdbc:mysql://localhost/your_database";

    // Database credentials
    String USER = "your_username";
    String PASS = "your_password";

    Connection conn = null;
    PreparedStatement stmt = null;

    try{
        // Register JDBC driver
        Class.forName(JDBC_DRIVER);

        // Open a connection
        conn = DriverManager.getConnection(DB_URL, USER, PASS);

        // Insert data into table
        String sql = "INSERT INTO your_table (first_name, last_name) VALUES
(?, ?)";
        stmt = conn.prepareStatement(sql);
        stmt.setString(1, firstName);
        stmt.setString(2, lastName);
        stmt.executeUpdate();

        // Display entered data
```

```
        out.println("<p>Entered Data:</p>");
        out.println("<p>First Name: " + firstName + "</p>");
        out.println("<p>Last Name: " + lastName + "</p>");

        // Clean-up environment
        stmt.close();
        conn.close();
    } catch(Exception e){
        // Handle errors
        out.println("Error: " + e.getMessage());
    }
%>
<form action="your_jsp_page.jsp" method="post">
    First Name: <input type="text" name="firstname"><br>
    Last Name: <input type="text" name="lastname"><br>
    <input type="submit" value="Submit">
</form>
</body>
</html>
```

Replace **"your_database"**, **"your_table"**, **"your_username"**, and **"your_password"** with your actual database details. Also, make sure to handle exceptions properly in a real-world application.