**1. Cos'è JDBC?**

* **Risposta**: JDBC (Java Database Connectivity) è un'API di Java che consente di collegare un'applicazione Java a un database, permettendo di eseguire query, recuperare dati e elaborare i risultati. JDBC facilita la comunicazione tra il programma Java e il database.

**2. Cosa serve per collegare un database MySQL a un programma Java?**

* **Risposta**: Per collegare un database MySQL a un programma Java, è necessario il **MySQL Connector/J**, un driver JDBC specifico per MySQL. Questo driver consente al programma Java di comunicare con MySQL. (Dove va inserito questo connettore?)

**3. Come si gestisce una connessione in JDBC?**

* **Risposta**: Per gestire una connessione in JDBC:
  + Si crea un'istanza di **Connection** usando una classe come ConnectionFactory.
  + Se la connessione è null, si crea una nuova connessione con il metodo DriverManager.getConnection, fornendo i dettagli di connessione (come il percorso al database e le credenziali).
  + È utile avere un metodo per chiudere la connessione quando non serve più, per liberare le risorse.

**4. Come appare il codice per gestire la connessione in JDBC?**

* **Risposta**: Ecco un esempio di codice per gestire la connessione:

|  |
| --- |
| **public** **class** ConnectionFactory {  **private** **static** Connection connection = null;  ​  **public** **static** Connection getConnection() {  **if** (connection == null) {  **try** {                 connection = DriverManager.getConnection(                         "jdbc:mysql://localhost:3306/nomeDatabase",                         "username",                         "password"                );            } **catch** (SQLException e) {                 e.printStackTrace();            }        }  **return** connection;    }  ​  **public** **static** void closeConnection() {  **if** (connection != null) {  **try** {                 connection.close();            } **catch** (SQLException e) {                 e.printStackTrace();            }        }    }  }  ​ |

**5. Cosa sono i PreparedStatement e perché sono importanti?**

* **Risposta**: I PreparedStatement sono un tipo di oggetto che permette di eseguire query SQL parametrizzate. Sono più sicuri rispetto a concatenare semplicemente stringhe per creare query, perché prevengono attacchi come la **SQL Injection**. Quando si usa PreparedStatement, i parametri della query vengono sostituiti con valori sicuri, rendendo la query meno vulnerabile.

**6. Come si usa un PreparedStatement per eseguire un inserimento di dati?**

* **Risposta**: Un esempio di inserimento dati con PreparedStatement:

|  |
| --- |
| String queryString = "INSERT INTO passeggero (nome, cognome, eta, numeroPassaporto) VALUES (?, ?, ?, ?)";  **try** (Connection conn = ConnectionFactory.getConnection();      PreparedStatement preparedStatement = conn.prepareStatement(queryString)) {  ​     preparedStatement.setString(1, passeggero.getNome()); // Primo "?"     preparedStatement.setString(2, passeggero.getCognome()); // Secondo "?"     preparedStatement.setInt(3, passeggero.getEta()); // Terzo "?"     preparedStatement.setString(4, passeggero.getNumeroPassaporto()); // Quarto "?"  ​     int updateRows = preparedStatement.executeUpdate();  ​  **if** (updateRows > 0) {         System.out.println("Inserimento avvenuto con successo!");    } **else** {         System.out.println("Nessun dato inserito.");    }  } **catch** (SQLException e) {     e.printStackTrace();  }  ​ |

**7. Cosa succede se la query non riesce a inserire i dati?**

* **Risposta**: Se updateRows > 0, significa che l’inserimento è avvenuto con successo, altrimenti non sono stati inseriti dati. Questo può essere indicato con un messaggio all’utente o con un log nel programma.

**8. Cos'è il ResultSet e a cosa serve?**

* **Risposta**: Il ResultSet è un oggetto che rappresenta il risultato di una query SQL di tipo SELECT. Quando una query restituisce dati (come una lista di righe), il ResultSet permette di scorrere i risultati e leggerli.

**9. Come funziona il metodo next() del ResultSet?**

* **Risposta**: Il metodo next() del ResultSet sposta il cursore alla riga successiva nei risultati della query. È comune usare un ciclo while con resultSet.next() per leggere tutte le righe restituite dalla query:

|  |
| --- |
| **while** (resultSet.next()) {     // Lettura delle colonne della riga corrente     String nome = resultSet.getString("nome");     String cognome = resultSet.getString("cognome");     // Altre colonne...  }  ​ |