Java.sql (esercitazione)

Insert con Java ConnectorJ ed java.sql

Dopo aver creato uno statemente, possiamo effettuare richiesta al Database con i metodi di execute:

```
boolean execute(String SQL);

// Questa è "la più conveniente" da utilizzare
// Restituisce informazioni più specifiche una volta eseguita
int executeUpdate(String SQL);


ResultSet executeQuery(String SQL);
```

Per avviare **xampp** da terminale:

```
sudo /opt/lampp/xampp start //(forse in futuro creerò un alias)
```

Esempio:

Inseriamo un nuovo veicolo nel database (esercizio Garage).

Ricorda: **avvia xampp prima**, se il progetto è nuovo **importare ConnectorJ.**

```
// Codice di una insert

    Connection connessione = null;
```

```
    try {

      // Mi collego al database
      connessione  = DriverManager.getConnection( urlDB, "root", "");

      // Creiamo lo statement
      Statement statement = connessione.createStatement();

      // Facciamo una richiesta al database e salviamo il risultato
      int risultato = statement.executeUpdate("insert into Veicolo values ('FF00FF','Sportiva','Maserati') ");

      // Controllo l'esito dell' insert
      if (risultato > 0) {
        System.out.println("success 200");
      }else{
        System.out.println("failure 400");
      }

      connessione.close();

    } catch (SQLException e) {

      // TODO Auto-generated catch block
      e.printStackTrace();

    }
```

Delete ed Update con Java ConnectorJ e java.sql

La Delete e la Update si fanno nello stesso modo, si cambia solo la richiesta al database:

```
// Facciamo una richiesta al database e salviamo il risultato
int risultato = statement.executeUpdate("delete from Veicolo where produttore = 'maserati' ");
```

Esercizio:

Modifichiamo la classe Garage ed implementiamo anche la delete ed update.

Soluzione:

ModelloModelloModello

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/1063f7ab-5b6b-4cb8-b3f4-2d86d9fc9b70/Update_Garage.zip

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/522278d6-f751-4776-abd8-b70d7502d464/Esercizio_Garage.zip

🤔 Piccolo Bug che ho trovato durante l'esercizio:

```
// Se il mio scan.nextLine() ogni tanto mi stampa cose inaspettate
// è perchè il Buffer è "pieno" o "sporco" dalle nextLine() fatte
// prima. Bisogna lanciare una nextLine() a vuoto per svuotare il Buff
// Come in C++ con  cin <<

scan.nextLine(); // A vuoto svuota il buffer
```

SQL Injection

Non si può risolvere con Java.

Passo in input alla delete la satringa:

where targa = 'A' OR 1=1 OR targa = 'A' → **mi cancella tutto il db**

Utilizzo di Prepared Statement

Fornisce un'interfaccia standard per la richiesta al DB, ossia una query "già pronta" in cui dobbiamo solo sostituire dei parametri.

Step:

1. Creiamo una **prepared Statement**.
2. La stinga da inserire, che rappresenta la richiesta, contiene **dei placeholder.**
3. Possiamo settare il valore dei **placeholder della query di prepared Statement.**
4. Ed infine procedere con **l’ execute della query.**

Una spiegazione molto tecnica ed esaustiva è qui:

https://www.geeksforgeeks.org/how-to-use-preparedstatement-in-java/#practice

```
// Esempio:

// Query which needs parameters -> OGNI ? è un placeHolder
String query = "Select * from students where age> ? and name = ?";

// Prepare Statement
PreparedStatement myStmt = con.prepareStatement(query);

// Set Parameters -> rimpiazzo il primo ? con 20 ed il secondo ? con
// 'Prateek'
myStmt.setInt(1, 20);
myStmt.setString(2, 'Prateek');

// Execute SQL query
ResultSet myRs = myStmt.executeQuery();
```

🤔 **Questa roba è utile quando:** vogliamo dare all’utente la capacità di ottenere informazioni dal database, e restituiamo un risultato d’accordo all’inserimento che lui esegue.

Si chiama “preparata” perchè viene eseguita “tante volte” dallo stesso utente che vuole reperire la stessa informazioni, ma con parametri diversi.

Ad esempio: Il professore che vuole sempre ottenere “lo studente x con matricola x” → vuole sempre uno Studente, ma con matricole diverse.

Esercizio

Chiediamo all’utente che cosa vuole fare e poi → Cosa vuole visualizzare.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/10322c14-5b9b-400e-a922-6b6e4c058549/Garage_preparedStatement.zip

HTML

Andiamo a creare i form per inserire / togliere nuove Auto nella nostra applicazione Garage.

Modello client-server

Client → fa la richiesta, è l’utente che fa una richiesta.

Server → riceve la richiesta, la gestisce ed invia una response (tutta la parte del backend)

DB → è la struttura dati d’appoggio del server, che mantiene le sue informazioni persistenti (storage di dati).

Esercizio

Creare in HTML + CSS il form di invio dei dati della classe Garage

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/e5a71dab-3181-4293-84df-22c2af0678f9/Garage_Interfaccia.zip