MyShopDB = MyShop + MySQL

- Adattiamo l'applicazione MyShop in modo da poter utilizzare un database come fonte di dati, invece dei soliti file testuali.
- I nuovi sorgenti si trovano nel file **`MyShopDB.zip`** scaricabile da:

  `www.dimi.uniud.it/scagnett/LabTecLS`

La base di dati (I)

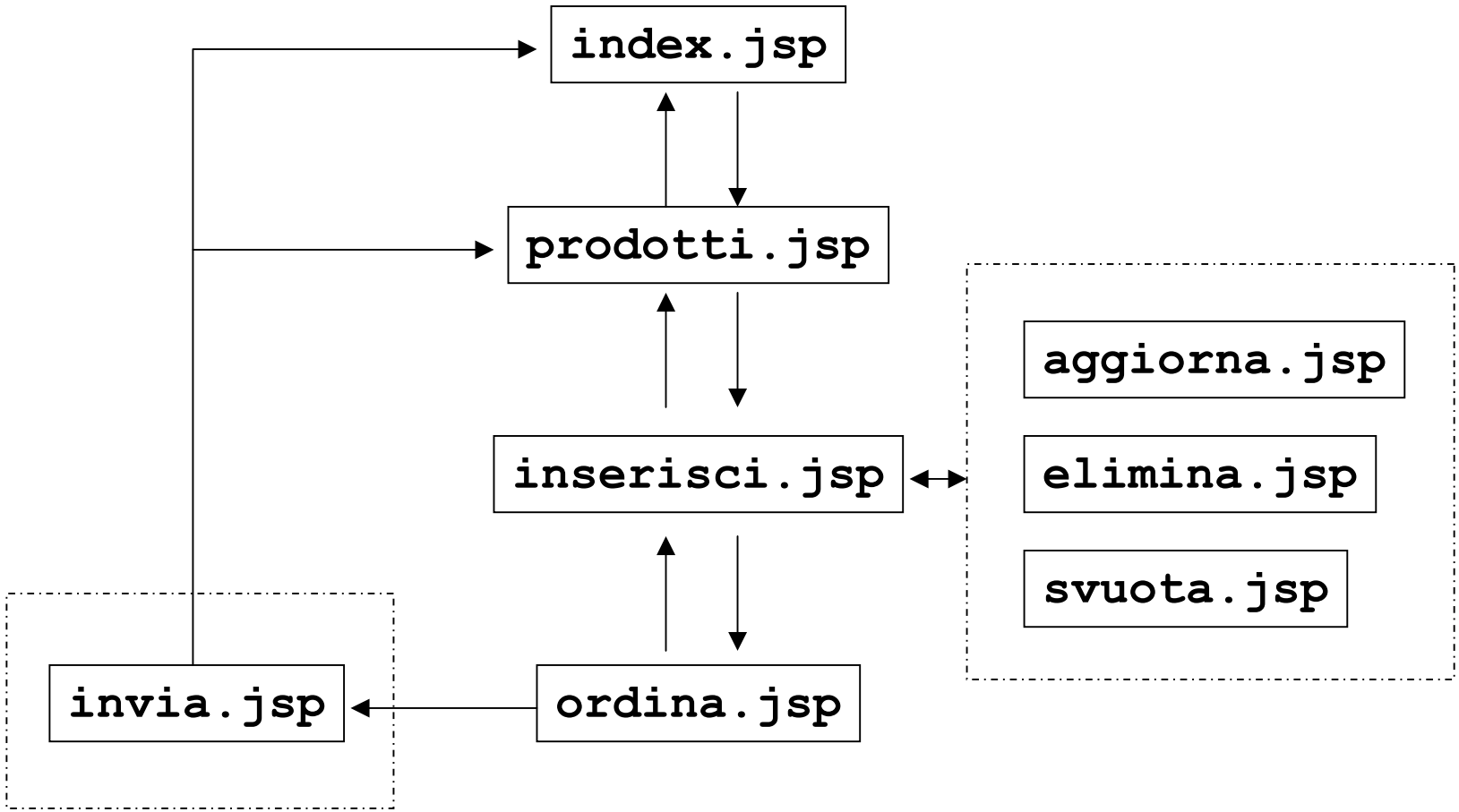
- Al database **`test_inventory`** su **`mizzi.dimi.uniud.it`** sono state aggiunte le seguenti tabelle:
  - **`prodotti`**,
  - **`carrelli`**,
  - **`intestazioni_ordini`**,
  - **`righe_ordini`**.

La base di dati (II)

- La tabella prodotti contiene i 3 elementi descritti nel file prodotti.txt della vecchia versione di MyShop:

```
[scagnetto@mizzi scagnetto]$ > mysql
mysql> use test_inventory
mysql> select * from prodotti;
+----+-----------------+-----------------------------------+--------+-----------
--------+
| ID | Prodotto        | Descrizione                       | Prezzo |
QuantitaMagazzino |
+----+-----------------+-----------------------------------+--------+-----------
--------+
|  1 | HD Seagate 80 GB | 7200 RPMS, Controller Serial Ata |     80 |
10 |
|  2 | Philips 107B     | Monitor 17&quot;, CRT             |    200 |
5 |
|  3 | RAM 512 MB       | Modulo SDRAM 512 MB (Kingston)   |    130 |
100 |
+----+-----------------+-----------------------------------+--------+-----------
--------+
3 rows in set (0.00 sec)
```

L'architettura di MyShopDB rimane invariata



File include

- Ai file utilizzati nelle direttive include se ne aggiunge uno nuovo:
  - **`sql.jsp`**
- In **`sql.jsp`** dichiariamo dei valori e delle funzioni utili per l'utilizzo di MySQL:
  - stringa di connessione,
  - nome utente e password per accedere a MySQL,
  - funzione per codificare gli apici nelle query SQL.

Il problema degli apici (I)

- Supponiamo di voler inserire un nuovo record nella tabella anagrafica (composta da tre campi: **`Nome`**, **`Cognome`**, **`DataNascita`**):

```
INSERT INTO anagrafica (Nome, Cognome, DataNascita) VALUES ('Mario', 'Rossi', '1987-10-05')
```

- Le stringhe di caratteri (come **`Mario`**) devono essere racchiuse fra apici.
- Quindi per inserire una stringa, al cui interno compaiono degli apici, come valore di un campo bisogna "codificarli" per evitare che l'interprete SQL "spezzi" la stringa in più parti con gli errori di sintassi che ne conseguirebbero.

Il problema degli apici (II)

- Gli apici in SQL vengono codificati raddoppiandoli:
  - es.: la stringa

    `l'articolo`

    viene codificata in

    `l''articolo`
- In questo modo è possibile delimitare la stringa con gli apici senza problemi nella sintassi SQL:

  `'l''articolo'`

Il problema degli apici (III)

- In Java è sufficiente scrivere la funzione seguente per la codifica:

```
String CodificaApici(String s) {
        String codifica="";
        for(int i=0; i<s.length(); i++)
                if(s.charAt(i)=='\'')
                        codifica+="''";
                else
                        codifica+=s.charAt(i);
        return codifica;
}
```

Il file `sql.jsp` (I)

```
<%@ page import="java.sql.*" %>
<%!
  String stringaConnessione="jdbc:mysql://mizzi.dimi.uniud.it/test_inventory";
  String utenteSQL="";
  String passwordSQL="";
  int lunghezzaMassima=100;
  double contributoSpeseSpedizione=30.0;

  String CodificaApici(String s) {
        String codifica="";
        for(int i=0; i<s.length(); i++)
                if(s.charAt(i)=='\'')
                        codifica+="''";
                else
                        codifica+=s.charAt(i);
        return codifica;
  }
%>
```

Parametri per la connessione al server MySQL

Il file `sql.jsp` (II)

```
...
<%
  try {
        Class.forName("com.mysql.jdbc.Driver");
  } catch (ClassNotFoundException e) {
        response.sendRedirect("errore.jsp");
  }
%>
```

La pagina **`errore.jsp`** visualizza un messaggio di avvertimento per l'utente.

Il file `prodotti.jsp` (I)

```
...
<%try {
    Connection c = DriverManager.getConnection(stringaConnessione,
                                               utenteSQL, passwordSQL);

    Statement s = c.createStatement();
    ResultSet r = s.executeQuery("SELECT * FROM prodotti");
    ResultSetMetaData md = r.getMetaData();%>
    ...
    <table border="0" cellpadding="5">
      <tr>
        <td class="intestazione_tabella">
          Prodotto
        </td>
        ...
        <td class="intestazione_tabella">
          Inserisci nel carrello
        </td>
      </tr>
```

Parametri definiti in **`sql.jsp`**

Il file `prodotti.jsp` (II)

```
<%int riga=0;
  while(r.next()) {
    riga++;%>
    <tr>
      <form method="post" action="inserisci.jsp">
      <%NumberFormat fmt=NumberFormat.getInstance(Locale.ITALIAN);
        fmt.setMinimumFractionDigits(2);
        fmt.setMaximumFractionDigits(2);
        int indicePrezzo=4; // indice del campo Prezzo
        int indiceQt=5;     // indice del campo QuantitaMagazzino
        for (int i = 2; i <= md.getColumnCount(); i++) {%>
          <td class='<%= riga%2==0 ? "riga2_tabella" :
                      "riga1_tabella"%>'>
            <% if(i==indicePrezzo)
                 out.print(fmt.format(r.getFloat(i)));
               else if(i==indiceQt)
                      out.print(r.getInt(i));
                    else
                      out.print(r.getString(i));%>
          </td>
        <%}%>
```

Il file `prodotti.jsp` (III)

```
  <td class='<%= riga%2==0 ? "riga2_tabella" : "riga1_tabella"%>'>
    <input type="text" name="qt" value="1" size="2">
  </td>
  <td class='<%= riga%2==0 ? "riga2_tabella" : "riga1_tabella"%>'>
    <input type="submit" name="ordina" value="&gt;&gt;">
    <input type="hidden" name="pid" value="<%= r.getInt(1)%>">
  </td>
  </form>
</tr>
<%}%>
</table>
<%r.close(); s.close(); c.close();
} catch (SQLException e) {
    response.sendRedirect("errore.jsp");
}%>
```

Il file `inserisci.jsp`

- Il codice della pagina risulta ora notevolmente semplificato; ad esempio:
  - controllo del carrello vuoto:

```
ResultSet r = s.executeQuery("SELECT COUNT(*) FROM carrelli WHERE
Utente='"+session.getId()+"'");
if(r.next()) {
   carrelloVuoto=r.getInt(1)==0;
}
```

  - controllo della presenza di un prodotto nel carrello:

```
r = s.executeQuery("SELECT COUNT(*) FROM carrelli WHERE
Utente='"+session.getId()+"' AND IDProdotto="+idProdotto);
if(r.next()) {
   prodottoNelCarrello=r.getInt(1)>0;
}
```

Il file `svuota.jsp`

```
<%@ include file="sql.jsp" %>
<%
  try {
    Connection c = DriverManager.getConnection(stringaConnessione,
                                                utenteSQL, passwordSQL);
    Statement s = c.createStatement();
    s.executeUpdate("DELETE FROM carrelli WHERE Utente='"+
                                                  session.getId()+"'");
    s.close();
    c.close();
  } catch (SQLException e) {
    response.sendRedirect("errore.jsp");
  }
  response.sendRedirect("inserisci.jsp?stampa=1");
%>
```

Il file `elimina.jsp` (I)

```
<%@ include file="sql.jsp" %>
<%String idProdotto=request.getParameter("id");
  int id=0;

  try {
        id = Integer.parseInt(idProdotto);
  } catch (NullPointerException e) {
        id=0;
  } catch (NumberFormatException e) {
        id=0;
  }
```

Controlliamo che l'id passato tramite il metodo GET del protocollo HTTP sia un intero.

E' importante fare questo controllo, in quanto un valore scorretto potrebbe generare un errore nell'esecuzione della query SQL (il metodo GET espone i parametri passati direttamente nell'URL).

Il file `elimina.jsp` (II)

```
if(id==0) {
      response.sendRedirect("errore.jsp");
}
try {
      Connection c = DriverManager.getConnection(stringaConnessione,
                                                 utenteSQL, passwordSQL);
      Statement s = c.createStatement();
      s.executeUpdate("DELETE FROM carrelli WHERE Utente='"
                  +session.getId()+"' AND IDProdotto="+idProdotto);
      s.close();
      c.close();
} catch (SQLException e) {
      response.sendRedirect("errore.jsp");
}
response.sendRedirect("inserisci.jsp?stampa=1");
%>
```

Il file `aggiorna.jsp` (I)

```
<%@ include file="sql.jsp" %>
<%int numeroRighe=0;
  try { numeroRighe = Integer.parseInt(request.getParameter("righe"));
  } catch (NullPointerException e) {
          numeroRighe=0;
  } catch (NumberFormatException e) {
          numeroRighe=0;
  }

  boolean carrelloVuoto=true;

  try {
    Connection c = DriverManager.getConnection(stringaConnessione,
                                               utenteSQL, passwordSQL);
    Statement s = c.createStatement();
```

Il file `aggiorna.jsp` (II)

```
for(int i=0;i<numeroRighe;i++) {
  String idProdotto=request.getParameter("id"+
                                    (new Integer(i+1)).toString());
  int qtProdotto=0;
  try {
    qtProdotto = Integer.parseInt(request.getParameter("qt"+
                                   (new Integer(i+1)).toString()));
  } catch (NullPointerException e) {
    qtProdotto=0;
  } catch (NumberFormatException e) {
    qtProdotto=0;
  }
```

Il file `aggiorna.jsp` (III)

```
  if(qtProdotto>0)
      s.executeUpdate("UPDATE carrelli SET QuantitaProdotto="+
            qtProdotto+" WHERE Utente='"+session.getId()+
            "' AND IDProdotto="+idProdotto);
  }

  s.close();
  c.close();
} catch (SQLException e) {
       response.sendRedirect("errore.jsp");
}

response.sendRedirect("inserisci.jsp?stampa=1");
%>
```

Il file **`invia.jsp`**

- E' presente la direttiva

  `<%@ page isThreadSafe="false" %>`
- Perché?
  - Il motivo risiede nel fatto che bisogna inserire un record nella tabella **`intestazioni_ordini`** ed uno o più record collegati dallo stesso **`ID`** nella tabella **`righe_ordini`**.
  - L'**`ID`** del nuovo ordine viene calcolato come il **`MAX(ID)`** dei vecchi ordini **`+1`**.
  - Quindi è necessario che soltanto un thread (richiesta) alla volta possano accedere al DB per calcolare correttamente l'ID del nuovo ordine (altrimenti potrebbero essere generati ID duplicati dalla pagina **`invia.jsp`**).

Esercizio

- Scrivere una pagina che visualizzi le intestazioni di ogni ordine (numero d'ordine, data, nome, cognome, indirizzo, pagamento, consegna, spese di spedizione).

  La query da utilizzare è:

```
SELECT ID, Data, Nome, Cognome,
Indirizzo, Pagamento, Consegna,
SpeseSpedizione FROM
intestazioni_ordini
```

Esercizio

- Oltre all'intestazione visualizzare le righe (corrispondenti ai prodotti ordinati) di ogni ordine, usando la query seguente (da ripetere per ogni singolo ordine):

```
SELECT P.Prodotto , R.Prezzo,
R.Quantita FROM prodotti AS P,
righe_ordini AS R WHERE
P.ID=R.IDProdotto AND
R.IDOrdine=<valore del campo ID
estratto usando la query precedente>
```